

#4  
S. Jones  
04/26/01  
JC971 U.S. PTO  
09/835623  
04/17/01

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

In re Patent Application of: )  
Hidetoshi IWASHITA )  
Application No.: To be Assigned ) Group Art Unit: To be Assigned  
Filed: April 17, 2001 ) Examiner: To be Assigned  
For: COMPILER SYSTEM, COMPILING METHOD, AND STORAGE MEDIUM FOR  
STORING COMPILING PROGRAM

**SUBMISSION OF CERTIFIED COPY OF PRIOR FOREIGN  
APPLICATION IN ACCORDANCE  
WITH THE REQUIREMENTS OF 37 C.F.R. §1.55**

*Assistant Commissioner for Patents  
Washington, D.C. 20231*

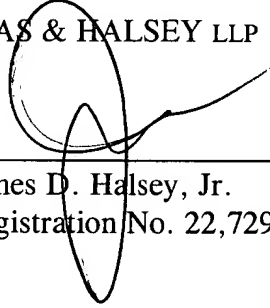
*Sir:*

In accordance with the provisions of 37 C.F.R. §1.55, the applicant(s) submit(s)  
herewith a certified copy of the following foreign application:

Japanese Patent Application No. PCT/JP99/00381  
Filed: January 29, 2001

It is respectfully requested that the applicant(s) be given the benefit of the foreign filing  
date as evidenced by the certified papers attached hereto, in accordance with the requirements  
of 35 U.S.C. §119.

Respectfully submitted,  
STAAS & HALSEY LLP

By:   
James D. Halsey, Jr.  
Registration No. 22,729

700 11th Street, N.W., Ste. 500  
Washington, D.C. 20001  
(202) 434-1500  
Date: April 17, 2001

## PATENT COOPERATION TREATY

Issuer: Japanese Patent Office (Receiving Office)

**Applicant Agent**

Yoshiyuki Osuga

**Address**

3rd Fl., Nibancho Bldg.  
8-20, Nibancho, Chiyoda-ku, Tokyo  
102-0084

**NOTICE OF MAILING-OUT  
THE REQUESTED DOCUMENT**

(The rule of the Law, Sec.37 & Sec.37-2)  
[PCT rule 20.9, 22.1 (d)]

**Date of mailing (day/month/year)**

13. 03. 01

**Applicant's or agent's file reference**

9805482

**Enclosure**

Official Copy of the International Application

**International application No.**

PCT/JP99/00381

**International filing date (day/month/year)**

29. 01. 99

**Applicant**

FUJITSU LIMITED

We, the Receiving Office, hereby confirm that "a set of the official copies of the documents associated with the international application at the time of the application and with its procedural supplements or its procedural amendments" as requested by the applicant is issued.

**Full address for the Receiving Office**

Japanese Patent Office (RO/JP)  
4-3, Kasumigaseki 3-chome  
Chiyoda-ku, Tokyo 100-8915  
Japan  
(phone No. 03-3592-1308)

**Authorized Officer**

Commissioner, Patent Office

We hereby certify that the accompanying tamper sealed copy  
represents the corresponding original documents.

Date: 12 March 2001

Seal of: Mr. Takayuki Kiyodera, Trade and Industry Officer

# 特 許 協 力 条 約

発信人 日本国特許庁（受理官庁）

出願人代理人

大菅 義之 殿

あて名

〒102-0084

東京都千代田区二番町8番地20

二番町ビル3F

請求があった書類の送付  
通知書

（法施行規則第37条、同第37条の2）

〔PCT規則20.9、22.1(d)〕

発送日（日．月．年）

13.03.01

出願人又は代理人の書類記号

9805482

国際出願の謄本又は写しの認証を同封

国際出願番号

PCT/JP99/00381

国際出願日（日．月．年）

29.01.99

出願人（氏名又は名称）

富士通株式会社

受理官庁は、出願人から請求のあった「出願時の国際出願に係わる書類又は手続の補完若しくは手続の補正に係わる書類の謄本」又は「~~国際出願の写しの認証~~」を交付する。

受理官庁の名称及びあて名

日本国特許庁（RO/JP）

郵便番号100-8915 TEL03-3592-1308

日本国東京都千代田区霞が関三丁目4番3号

様式PCT/RO/122（1992年7月）

権限のある職員

特 許 庁 長 官

受理官庁用写し

特許協力条約に基づく国際出願

願 書

出願人は、この国際出願が特許協力条約に従って処理されることを請求する。

国際出願番号

PCT/JP99/00381

国際出願日

29.01.99

(受付印)

PCT International Application  
日本国特許庁

出願人又は代理人の書類記号  
(希望する場合、最大12字)

9805482

第 I 欄 発明の名称

コンパイラ装置、コンパイル方法、およびそのためのプログラムを格納した記憶媒体

第 II 欄 出願人

氏名(名称)及びあて名:(姓・名の順に記載;法人は公式の完全な名称を記載;あて名は郵便番号及び国名も記載)

富士通株式会社

FUJITSU LIMITED

〒211-8588 日本国神奈川県川崎市中原区上小田中4丁目1番1号

1-1, Kamikodanaka 4-chome, Nakahara-ku, Kawasaki-shi,

Kanagawa 211-8588, Japan

☐ この欄に記載した者は、  
発明者でもある。

電話番号:

044-754-3034

ファクシミリ番号:

044-754-3563

加入電話番号:

国籍(国名): 日本国 JAPAN

住所(国名): 日本国 JAPAN

この欄に記載した者は、次の

指定国についての出願人である:

☐

すべての指定国

☒

米国を除くすべての指定国

☐

米国のみ

☐

追記欄に記載した指定国

第 III 欄 その他の出願人又は発明者

氏名(名称)及びあて名:(姓・名の順に記載;法人は公式の完全な名称を記載;あて名は郵便番号及び国名も記載)

岩下 英俊 IWASHITA Hidetoshi

〒211-8588 日本国神奈川県川崎市中原区上小田中4丁目1番1号

富士通株式会社内

c/o FUJITSU LIMITED, 1-1, Kamikodanaka 4-chome, Nakahara-ku,

Kawasaki-shi, Kanagawa 211-8588, Japan

この欄に記載した者は  
次に該当する:

☐

出願人のみである。

☒

出願人及び発明者である。

☐

発明者のみである。  
(ここにレ印を付したとき  
は、以下に記入しないこと)

国籍(国名): 日本国 JAPAN

住所(国名): 日本国 JAPAN

この欄に記載した者は、次の

指定国についての出願人である:

☐

すべての指定国

☐

米国を除くすべての指定国

☒

米国のみ

☐

追記欄に記載した指定国

☐ その他の出願人又は発明者が続表に記載されている。

第 IV 欄 代理人又は共通の代表者、通知のあて名

次に記載された者は、国際機関において出願人のために行動する:

☒

代理人

☐

共通の代表者

氏名(名称)及びあて名:(姓・名の順に記載;法人は公式の完全な名称を記載;あて名は郵便番号及び国名も記載)

7409 弁理士 大 菅 義之 OSUGA Yoshiyuki

〒102-0084 日本国東京都千代田区二番町8番地20 二番町ビル3F

3rd Fl., Nibancho Bldg., 8-20 Nibancho, Chiyoda-ku,

Tokyo 102-0084, JAPAN

電話番号:

03-3238-0031

ファクシミリ番号:

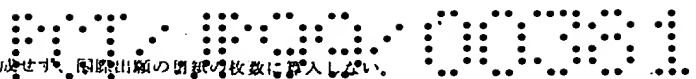
03-3238-0034

加入電話番号:

☐ 通知のためのあて名: 代理人又は共通の代表者が選任されておらず、上記枠内に特に通知が送付されるあて名を記載している場合は、レ印を付す







P C T

手 数 料 算 用 紙

願 書 附 属 書

受理官庁記入欄

国際出願番号

受理官庁の日付印

出願人又は代理人の書類記号

9805482

出願人

富士通株式会社

所定の手数料の計算

1. 及び 2. 特許協力条約に基づく国際出願等に関する法律（国内法）  
第18条第1項第1号の規定による手数料（注1）  
（送付手数料 [T] 及び調査手数料 [S] の合計）

95,000.- 円 T+S

3. 国際手数料（注2）

基本手数料

国際出願に含まれる用紙の枚数 73 枚

最初の30枚まで

62,800.- 円 b1

43 × 1,450 =

62,350.- 円 b2

30枚を超える用紙の枚数 用紙1枚の手数料

b1及びb2に記入した金額を加算し、合計額をBに記入

125,150.- 円 B

指定手数料

国際出願に含まれる指定数（注3） 2

2 × 14,500 =

29,000.- 円 D

支払うべき指定手数料  
の数（上限は11）  
（注4）

1指定当たりの手数料  
（円）

B及びDに記入した金額を加算し、合計額をIに記入

154,150.- 円 I

4. 納付すべき手数料の合計

T+S及びIに記入した金額を加算し、合計額を合計に記入

249,150.- 円

合 計

（注1）送付手数料及び調査手数料については、合計金額を特許印紙をもって納付しなければならない。

（注2）国際手数料については、受理官庁である日本国特許庁の長官が告示する国際事務局の口座への振込みを証明する書面を提出することにより納付しなければならない。

（注3）願書第V欄でレ印を付した口の数。

（注4）指定数を記入する。ただし、11指定以上は一律11とする。

95,000/-

62,800/-

62,350/-

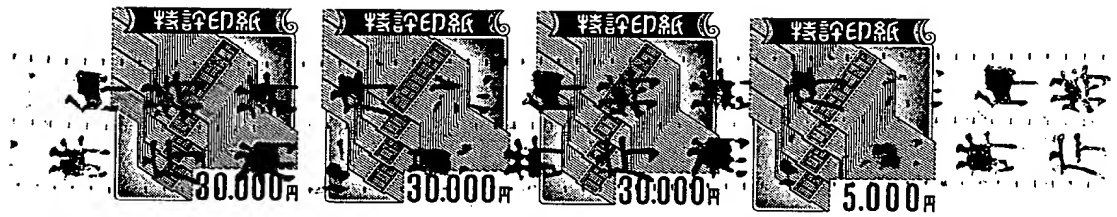
125,150/-

29,000/-

154,150/-

249,150/-





送付手数料・調査手数料 95,000 円

# 振込金受取書

電信扱

文書扱

さくら銀行

(兼手数料受取書)

さくら銀行をご利用いただきありがとうございました。

①次回より早くて便利なATMをご利用いただきますと、手数料がお得になります。

②午後2時以降は、窓口が大変混雑いたしますので、お振込は出来るだけ早めにご依頼ください。

振込 種別	11 年 1 月 29 日	③振込額が あつてくださ
振込 先	東京三菱 内幸町	金 額
振込 種別	①通 2. 当 4. 貯 9.	0473286
フリガナ	おなまえ	
おなまえ	WIPO - PCT GENEVA	様
フリガナ	おてんわ	
おなまえ	オオスガナイガイコトキョジムショ	
おなまえ	大菅内外国特許事務所	
おところ	オオ スガ ヨシ ユキ	様
おところ	大 菅 義 之	
〒102 東京都千代田区二番町8-20 二番町ビル3階		
ご連絡先おて 電話3238-0031 FAX3238-0034		

83	吉田
----	----

株式会社さくら銀行

振込依頼書に記載相違等があった場合には、照会等のため振込が遅延したり、振込ができないことがあります。  
通信機器、回線の障害またはその他の遅延等やむを得ない事由によって、振込が遅延することがありますのでご了承ください。  
この振込金受取書は、お振込の金額などに必要となりますので、ご依頼人が大切に保管してください。  
文書扱は、お振込の金額に誤りがありますからご承知おきください。



基本手数料	125,150 円
指定手数料	29,000 円
合計	154,150 円

2001.99

包 括 委 任 状

平成 8 年 8 月 27日

私儀 弁理士 大菅義之 氏  
を代理人と定めて下記の権限を委任します。

1. 特許協力条約に基づくすべての国際出願に関する一切の件
2. 上記出願又は指定国の指定を取り下げる件
3. 上記出願に対する国際予備審査の請求に関する一切の件並びに  
選択国の選択を取り下げる件

あて名 〒211 日本国神奈川県川崎市中原区上小田中4丁目1番1号  
名 称 富 士 通 株 式 会 社  
代表取締役社長 関澤 義



## 明 細 書

コンパイラ装置、コンパイル方法、

およびそのためのプログラムを格納した記憶媒体

5

## 技術分野

本発明は、ある言語で記述された入力プログラムの少なくとも一部をコンパイルしてその言語と同一の言語またはその言語と異なる他の言語で記述されたプログラムを出力するコンパイラに係わる。特に、出力プログラムが Fortran  
10 またはC言語などであるプリプロセッサ方式のコンパイラに係わる。

## 背景技術

コンピュータの動作を記述するためのプログラミング言語は、使用目的や適用分野などに応じて様々な拡張言語または改良言語が開発されている。たとえば、Fortran をベースとする拡張言語としては、HPFが広く知られている。  
15 また、C言語をベースとする拡張言語としては、C++、C9x、DPC Eなどが知られている。

ところが、これらの拡張言語で記述されたプログラム等をアセンブラまたは機械語に翻訳する際には、いったんベース言語に変換することが有利なことが  
20 多い。このため、これらの拡張言語を扱うコンパイラは、一般に、出力の言語がその拡張言語のベース言語であるプリプロセッサ方式で実現されることが多い。たとえば、HPFで記述されたプログラムをFortranで記述されたプログラムに変換して出力するコンパイラが開発されている。

関数 SUM\_SCATTER は、下記の形式で呼び出される。

SUM\_SCATTER (ARRAY, BASE, INDX1, INDX2, ... , INDXn, MASK)

- ここで、「ARRAY」は、任意の数値型であり、且つ任意の次元数を持つ配列である。数値型は、処理系が許す任意のバイト数の整数型、実数型、複素数型などである。「BASE」は、「ARRAY」と同じ型であり、任意の次元数を持つ配列である。「INDX1, INDX2, ... , INDXn」は、スカラであるか「BASE」と同じ次元数を持つ配列であり、「n」は「ARRAY」と同じ次元数である。「MASK」は、スカラであるか、「ARRAY」と同じ次元数を持つ配列であるか、あるいは省略される。

したがって、たとえば、もし、処理系が10種類の数値型を持ち、1次元～7次元までの配列をサポートすると仮定すると、この関数の自由度は以下のよう計算できる。

- 「ARRAY」が1次元のときは、「ARRAY」の自由度が10、「BASE」の自由度が7、「n=1」なので「INDX1, INDX2, ... , INDXn」の自由度が2、さらに「MASK」の自由度が3である。従って、この場合、組合せの数は、 $10 \times 7 \times 2 \times 3 = 420$ となる。同様に、「ARRAY」が2次元～7次元のときの組合せの数は以下の通りである。

$$\begin{aligned} 2 \text{次元} : & 10 \times 7 \times 2^2 \times 3 = 840 \\ 3 \text{次元} : & 10 \times 7 \times 2^3 \times 3 = 1680 \end{aligned}$$

$$7 \text{次元} : 10 \times 7 \times 2^7 \times 3 = 26880$$

よって、合計の組合せ数は、53340通りになる。さらに、HPFでは、一般に、配列は複数のプロセッサに分割して配置されるので、分割配置の種類 (block, cyclic など) や分割次元の違い、および多次元分割などを考慮す

5    ると、その組合せは膨大なものとなる。

したがって、抽象度の高い組込み手続の変換を、通常ライブラリを用いて実現しようとする、以下の問題が生じる。

(1) すべての組合せに対応しようとする、ライブラリが巨大になり、結果としてそのライブラリを記憶するための記憶媒体 (メモリ容量) も大きくなっ  
10    てしまう。

(2) 複数のケースに対応できるような入口名 (総称的な名前、あるいは抽象的な名前) を使ったとしても、結局はライブラリの中でケース分けをする必要がある、上記(1)の問題は解決されない。

(3) 複数のケースに対応できるような入口名 (総称的な名前、あるいは抽象  
15    的な名前) を使い、且つ様々な型や次元数でライブラリを作る方法も考えられるが、実際には以下の問題を伴う。

(a) 次元数の違いに対応するためには、ループのネスト数を可変にできればよいが、Fortran やC言語ではそのような記述はできない。また、再帰呼出しで記述すると、実行速度が低下する。

20    (b) 型 (たとえば、4バイトの整数型や8バイトの実数型) を共通化することは困難である。たとえば、プログラム中の式の中に整数型の数値と実数型の数値が混在していた場合、これらの型は実行コードが互いに異なるので、型ごとに処理を分ける必要がある。このため、命令列中に判断分岐が多数登場す

ることになり、実行速度が低下する。

(c) 複数のプロセッサに分割された配列を引数としてサポートする言語においては、単純な分割（例えば、最終次元での均等な block 分割）に対しても一般的な分割（すべての次元において任意の分割方法で分割できる方法を想定

5 する）と同じ扱いになるので、実行速度が低下してしまう。

このように、抽象度の高い組込み手続をプリプロセッサ方式で変換しようとする  
と、ライブラリを記憶するための記憶媒体が巨大になってしまった。あるいは、記憶媒体の巨大化を回避しようとする、コンパイル能力が低下していた。

10 この問題を回避する方法の 1 つとして、組込み手続のすべての部分についてライブラリを用意しておくのではなく、組込み手続の呼出しパラメータに係わる部分のみをコンパイル時に生成（具体化）する方法が考えられる。例えば、インライン展開技術を用いてソースプログラム中の手続き呼出を変換する手法が考えられる。

15 ところが、プリプロセッサ方式のコンパイラにおいては、手続き呼出の登場場所によっては、インライン展開が出来なかったり、困難であったりすることがある。以下、問題点を説明する。

(1) 通常、宣言部において実行文を展開することは出来ない。このため、宣言部に関数呼出等が存在する場合は、それをインライン展開することはできない。もし、宣言部に登場する関数呼出等をそのままインライン展開すると、得られたプログラムをベース言語のコンパイラで翻訳したときにエラーが発生することになる。たとえば、図 1 A に示す入力プログラムの宣言部を Fortran プログラムにコンパイルすると、図 1 B に示すコードが得られる。また、図 2 A

20





- このように、プログラム等をコンパイルする際、インライン展開が可能であれば、そのコンパイルに必要な情報を格納するための記憶領域を小さくすることができるなどのメリットが得られるが、オブジェクト言語が処理手続を記述できる場所について制約がある言語であった場合には、インライン展開が出来なかつたり、あるいは困難だつたりする。すなわち、オブジェクト言語が機械語やアセンブラのように実行コードの出現場所についての制約が少ないコンパイラにおいては、インライン展開をするにあたっての問題も少ないが、オブジェクト言語が Fortran や C 言語などのように実行コードの出現場所についての制約が多いプリプロセッサ方式のコンパイラにおいては、インライン展開をするにあたって上述のような問題が発生しやすい。
- 5
- 10

本発明の課題は、実行速度が高速で、且つコンパイルに必要な情報を格納するための記憶領域が小さくなるプリプロセッサ方式のコンパイラを提供することである。

## 15 発明の開示

- 本発明のコンパイラ装置は、入力された第 1 のプログラムをコンパイルして第 2 のプログラムを出力する構成を前提とし、第 1 のプログラムの一部を手続呼出に変換して上記第 2 のプログラムを作成する変換ユニットと、上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成する生成ユニットと、上記第 2 のプログラムおよび上記生成手段により生成されたコードを出力する出力ユニットとを有する。
- 20

第 1 のプログラムをコンパイルすることにより第 2 のプログラムを作成する際、手続の定義を記述したコードは第 2 のプログラムの外に生成され、第 2 の

プログラム中にはそのコードを呼び出すための手続呼出が記述される。すなわち、第2のプログラム中には実行コードは展開されない。したがって、オブジェクト言語が処理手続を記述できる場所について制約がある言語であった場合においても、実行されたときにエラーの発生を伴わない第2のプログラムが得られる。また、第2のプログラムの記述が複雑になることはなく、インライン展開を用いる場合と比べ、第2のプログラムをベース言語コンパイラで翻訳する速度の向上が期待される。さらに、上記装置においては、入力プログラム中の変換すべき対象のすべての組合せに対してコンパイルに必要な情報を用意しておく必要がないので、そのためのメモリ容量が小さくなる。

10

#### 図面の簡単な説明

図1Aおよび図1Bは、従来のコンパイラによってインライン展開できない例（その1）である。

15

図2Aおよび図2Bは、従来のコンパイラによってインライン展開できない例（その2）である。

図3Aおよび図3Bは、従来のコンパイラによってインライン展開が複雑になる例である。

図4は、本実施例のコンパイラの概略動作の一例を説明する図である。

図5は、本実施例のコンパイラの機能ブロック図である。

20

図6は、図5に示したコンパイラの動作を説明するためのフローチャートである。

図7Aは、図5に示したコンパイラへの入力コードの例である。

図7Bは、図5に示したコンパイラからの出力コードの例である。

図 8 は、関数 SUM を展開するためのテンプレートの例である。

図 9 A および図 9 B は、テンプレートを用いて関数 SUM を展開した結果を示す図である。

図 10 は、本発明の他の形態のコンパイラの機能ブロック図である。

5 図 11 は、図 10 に示したコンパイラの動作を説明するためのフローチャートである。

図 12 A は、図 10 に示したコンパイラへの入力コードの例である。

図 12 B は、図 10 に示したコンパイラからの出力コードの例である。

10 図 13 A ～図 13 D は、重複判定において使用する管理テーブルを説明する図である。

図 14 は、関数 SUM を展開した結果を示す図である。

図 15 は、抽象度の高い表現で作成したテンプレートの例である。

図 16 は、図 15 に示すテンプレートを用いて変換すべき対象を展開する際に比較されるパラメータを示す図である。

15 図 17 は、図 15 に示すテンプレートを用いた場合のコンパイラの出力を示す図である。

図 18 は、本発明の更に他の形態のコンパイラの機能ブロック図である。

図 19 は、図 18 に示したコンパイラの動作を説明するためのフローチャートである。

20 図 20 は、図 18 に示したコンパイラへの入力コードの例である。

図 21 は、図 18 に示したコンパイラからの出力コードの例である。

図 22 は、本発明の更に他の形態のコンパイラの機能ブロック図である。

図 23 は、図 22 に示したコンパイラの動作を説明するためのフローチャー

トである。

図 2 4 は、図 2 2 に示したコンパイラへの入力コードの例である。

図 2 5 は、図 2 2 に示したコンパイラからの出力コードの例である。

図 2 6 A は、入力コードの例である。

5 図 2 6 B は、図 2 6 A に示す入力コードに対応する出力の例である。

図 2 7 A ～図 2 7 D、図 2 8 A、および図 2 8 B は、テンプレートの例である。

図 2 9 は、本発明に係わるソフトウェアプログラムを実行するコンピュータのブロック図である。

10 図 3 0 は、本発明に係わるソフトウェアプログラムの提供方法を説明する図である。

#### 発明を実施するための最良の形態

本発明は、入力されたプログラムの少なくとも一部をコンパイルするプリプロセッサ方式のコンパイラに係わる。プリプロセッサ方式のコンパイラ（「プリコンパイラ」とも呼ばれている）は、ソースプログラムを読み込み、そのソースプログラムの少なくとも一部を変更してそのソースプログラムのコンパイルのための準備を行うプログラムである。以下の実施例では、Fortran および C 言語を採り上げ、ソースプログラムの中の呼出（関数、サブルーチン、ライブラリ等の組込み手続を呼び出すための呼出）などをオブジェクトコードに変換する機能を中心に説明する。なお、本実施例のコンパイラは、ソース言語とオブジェクト言語とが互いに異なっている場合、およびソース言語とオブジェクト言語とが互いに同じ場合の双方に適用可能である。

15

20

本実施例のコンパイラの概略動作の一例を図4を参照しながら説明する。ここでは、ソースプログラム1が、たとえば、Fortranの拡張言語の1つであるHPFで記述されており、その中に「A\*\*N」が含まれているものとする。

「A\*\*N」は、「AのN乗」を表す。また、オブジェクト言語は、例えば、  
5 Fortranであるものとする。この場合、コンパイラ2は、たとえば、ソースプログラム1の中の「A\*\*N」を「POW(A, N)」に変換することによりオブジェクトプログラム3を生成すると共に、「POW(A, N)」により呼び出されるべき手続を記述するコード(オンラインコード)4を出力する。この手続の定義は、たとえば、「Nが所定値よりも小さいときには掛け算を実行  
10 し、Nが所定値以上のときには巾乗演算を実行する。」である。また、コード4は、この手続をオンライン展開することにより得られる。なお、「オンライン展開」とは、あるプログラム中に登場する手続等をそのプログラムの外でサブプログラムとして展開することを意味する。また、オンライン展開により得られるコードを「オンラインコード」と呼ぶことにする。

15 なお、コンパイラ2は、必要に応じて、オブジェクトプログラム3の手続呼出を用いてオンラインコード4を呼び出すための記述(たとえば、インタフェース宣言)をオブジェクトプログラム3に加える。

このように、コンパイラ2は、ソースプログラム1の一部を手続呼出に変換することによりオブジェクトプログラム3を生成する際、その手続呼出により  
20 呼び出されるべきコードをオンラインコード4としてオブジェクトプログラム3の外に作成する。すなわち、手続呼出により呼び出されるべきコードは、オブジェクトプログラム3の中では展開されない。従って、オブジェクト言語が手続を記述できる場所について制約がある言語であったとしても、オブジェク

トプログラム 3 が実行されたときに文法エラー等が発生することはない。

図 5 は、本実施例のコンパイラの機能ブロック図である。ここでは、プログラム毎にソースコードが与えられることを前提とする。

ソースプログラムは、構文解析部 21 により分解・解析された後にコンパイラ 10 に入力される。構文解析部 21 は、入力コードを中間コードに変換する機能を備え、既存のプログラムをそのまま流用できる。

コンパイラ 10 は、検出部 11、変換部 12、および展開部 13 を備える。検出部 11 は、変換すべき対象を検出する。具体的には、検出部 11 は、ソースプログラムから予め決められた特定のパターン（たとえば、手続呼出など）を検出する。変換部 12 は、検出部 11 により検出された変換すべき対象に対応する手続名を決定し、その変換すべき対象をその手続名に置き換える。そして、コンパイラ 10 は、変換部 12 によりその一部が変換されたソースプログラムをオブジェクトプログラムとして出力する。展開部 13 は、検出部 11 により検出された変換すべき対象に対応するオンラインコード（手続コードまたは実行コード）を生成する。なお、このオンラインコードは、オブジェクトプログラムが実行されたときに、変換部 12 により決定された手続名に対応付けられて呼び出される。そして、コンパイラ 10 は、その展開部 13 により生成されたオンラインコードを出力する。

コード生成部 22 は、コンパイラ 10 から出力されたオブジェクトプログラムおよびオンラインコードをアセンブラまたは機械語に翻訳し、それらをそれぞれ出力する。コード生成部 22 は、既存のプログラムをそのまま流用でき、最適化処理を行うこともできる。

図 6 は、図 5 に示したコンパイラ 10 の動作を説明するためのフローチャー

トである。ここでは、コンパイラ10に入力されるソースプログラムをプログラムP、コンパイラ10によりコンパイルされたオブジェクトプログラムをプログラムP'と呼ぶことにする。なお、この処理は、1つのソースプログラムが入力される毎に実行される。

- 5     ステップS1では、プログラムP内で変換すべき対象C1, ..., Cnを検出する。変換すべき対象C1, ..., Cnは、たとえば、関数、サブルーチン、ライブラリの呼出である。

ステップS2~S5の処理は、ステップS1において検出された各変換すべき対象C1, ..., Cnについて実行される。ステップS2では、変換すべき対象Ciの特徴Aiを抽出する。「特徴」とは、例えば、関数名などの呼び出すべき手続の名前、引数の次元、数値型などである。ステップS3では、変換すべき対象Ciに対して、プログラムP内でユニークな手続名fiを生成する。すなわち、各変換すべき対象C1, ..., Cnに対して、互いに異なる手続名を割り当てる。

- 15     ステップS4では、変換すべき対象Ciを手続fiの呼出に置換する。ここで、「手続fiの呼出」とは、当該プログラムが実行されたときに、手続fiを呼び出すための呼出である。ステップS5では、特徴Aiに対応する手続コードSiを生成する。そして、この手続コードSiの手続名としてfiを指定する。これにより、手続コードSiは、当該プログラムが実行されたときに、  
20     手続fiの呼出により呼び出されることになる。

上記ステップS2~S5の処理をすべての変換すべき対象C1, ..., Cnについて実行すると、プログラムPにおいて各変換すべき対象C1, ..., Cnがそれぞれ手続f1, ..., fnの呼出に置換され、また、各手続f1, ..., fn

の呼出にそれぞれ対応して手続コード  $S_1, \dots, S_n$  が生成される。そして、ステップ S 6 において、変換すべき対象  $C_i$  を手続  $f_i$  の呼出に置き換えることにより得られるプログラム  $P'$ 、及び生成された手続コード  $S_1, \dots, S_n$  が出力される。なお、プログラム  $P'$  および手続コード  $S_1, \dots, S_n$  は、互

5 いに同じファイルに出力されてもよいし、互いに異なるファイルに出力されてもよい。また、ソースコードが複数のプログラム単位を含む場合には、各プログラム単位に対して上記処理が繰り返し実行される。

次に、具体的な実施例を説明する。以下では、コンパイラ 10 に対して図 7 A に示すソースプログラムが入力された場合を想定し、関数 SUM の呼出を変換

10 すべき対象として説明する。なお、関数 SUM は、引数により指定された配列の要素の総和を返す関数であるものとする。また、ソースプログラムは、中間コードに変換されていてもよい。

ソースプログラムが入力されると、コンパイラ 10 は、そのソースプログラムをスキャンすることにより関数 SUM の呼出を検出する。図 7 A に示す例にお

15 いては、関数 SUM は、4 行目の「SUM(A)」および 5 行目の「SUM(N(51:100))」により呼ばれていることが検出される。

続いて、コンパイラ 10 は、「SUM(A)」をユニークな名前が割り当てられた手続の呼出に変換する。この実施例では、新たに生成される名前は、変換すべき対象の手続名と、ソースプログラムの名前と、そのソースプログラムの中で

20 登場した順番との組合せにより決定される。この場合、図 7 A に示す例では、「SUM(A)」はプログラム SAMPLE の中で最初に登場した変換対象なので、新たな手続名として「SUM\_SAMPLE\_1」が割り当てられる。なお、コンパイラ 10 が新たに生成する名前とユーザにより定義される変数名などとの重複を回避する



ためには、コンパイラ 10 は、ユーザが使用できない名前を選択するようにすればよい。たとえば、「xxx\_」で始まる名前を予め予約しておくことによりユーザの使用を制限する方法や、「#」や「\$」などのユーザが滅多に使用しない文字を含む名前を生成するようにする方法が考えられる。

- 5      この後、コンパイラ 10 は、ソースプログラムの 4 行目を以下のように変換する。このとき、「SUM\_SAMPLE\_1」の引数は、「SUM(A)」の引数をそのまま使用する。

B=SUM(A)    →    B=SUM\_SAMPLE\_1(A)

- 10      続いて、コンパイラ 10 は、引数 A を解析することにより、「SUM(A)」の特徴を抽出する。これにより、以下の情報が得られる。

arg-type    =    REAL

m = 3

lb(1) = 1

ub(1) = 10

- 15      lb(2) = 1

ub(2) = 20

lb(3) = 1

ub(3) = 30

- 20      ここで、「arg-type」は引数の型、「m」は引数の次元数、「lb(i)」および「ub(i)」はそれぞれ引数の第 i 次元の下限值および上限値である。

コンパイラ 10 は、「SUM\_SAMPLE\_1」の手続コードを生成する際、テンプレートを利用する。テンプレートは、予め決められた変換すべき対象を展開するためのひな形であり、データベース等に格納されている。関数 SUM を展開する

ためのテンプレートの例を図8に示す。

テンプレートは、抽象的な表現を用いて作成されており、変換すべき対象の特徴に応じてそのテンプレートをカスタマイズすることにより対応する手続コードが生成される。また、1つの呼出に対して複数のテンプレートを設けてもよい。例えば、重要な呼出に対しては複数種類のテンプレートを設けておき、その呼出の特徴に応じてそれら複数のテンプレートの中から適切な1つを選択して利用するようにしてもよい。さらに、手続の引数などをパラメータとして具体的なソースコードまたはオブジェクトコードを生成するような処理プログラムとしてテンプレートを作成することも可能である。

- 10 コンパイラ10は、図8に示すテンプレートを利用し、引数Aを解析することにより得られている情報に従って「SUM\_SAMPLE\_1(A)」についての手続コードを生成する。このようにして生成された手続コードの例を図9Aに示す。

- 手続コードは、具体的には、以下のようにして作成される。すなわち、引数Aの「型」が「REAL」であるので、図8に示すテンプレートにおいて、「arg-type」として「REAL」が書き込まれる。また、引数の各次元の下限值および上限値として「1」「10」「1」「20」「1」「30」が書き込まれる。さらに、このコードが「SUM\_SAMPLE\_1(A)」により呼び出されるようにするために、手続名として「SUM\_SAMPLE\_1」が書き込まれる。

- 「SUM(N(51:100))」についての処理は、基本的に、「SUM(A)」についての処理と同じである。すなわち、コンパイラ10は、まず「SUM(N(51:100))」をユニークな名前が割り当てられた手続の呼出に変換する。図7Aに示す例では、「SUM(N(51:100))」はプログラムSAMPLEの中で2番目に登場した関数SUMなので、新たな手続名として「SUM\_SAMPLE\_2」が割り当てられる。この後、コン

パイラ 10 は、ソースプログラムの 5 行目を以下のように変換する。

WRITE(\*,\*) SUM(N(51:100)) → WRITE(\*,\*) SUM\_SAMPLE\_2(N(51:100))

続いて、コンパイラ 10 は、引数を解析して「SUM(N(51:100))」の特徴を抽出する。これにより、以下の情報が得られる。

5     arg-type = INTEGER

      m=1

      lb(1) = 51

      ub(1) = 100

この後、コンパイラ 10 は、図 8 に示したテンプレートを利用して「SUM\_SAMPLE\_2(N(51:100))」についての手続コードを生成する。生成された手続コードの例を図 9 B に示す。

図 7 B は、コンパイラ 10 を用いて図 7 A に示したソースプログラムをコンパイルした結果を示す図であり、図 6 に示したフローチャートにおけるプログラム P' に相当する。図 7 A および図 7 B に示すように、ソースプログラム中に複数の変換すべき対象（関数 SUM を呼び出すための呼出）が存在する場合、  
15     コンパイラ 10 は、それらに対して互いに異なる手続名を割り当てることにより、それらの変換対象を互いに識別可能な手続呼出に置き換える。この変換処理により得られたコード（オブジェクトプログラム）は、図 7 B に示すようなプログラムコードイメージか、対応するオブジェクトコードとしてファイルに  
20     出力される。また、コンパイラ 10 は、各手続呼出に対して、手続コードを生成して出力する。ここで、コンパイラ 10 は、複数の手続コードを一括して出力してもよいし、生成するごとに、順次、出力するようにしてもよい。

このように、コンパイラ 10 は、図 7 A に示すソースプログラムが入力され

ると、図7Bに示すオブジェクトコード、および図9Aおよび図9Bに示す手続コードを出力する。ここで、手続コードは、オブジェクトコード内では展開されない。したがって、コンパイラ10のオブジェクト言語が手続コードを記述できる場所について制約を持っていたとしても、コンパイル結果がその制約に違反することはない。

従来、たとえば、FORTRAN やC言語などでは、プログラムの宣言部において実行文を展開することができないので、宣言部に組み込み手続等の呼出が記述されていた場合には、従来のコンパイラではその宣言部を正しくコンパイルすることはできなかった。ところが、本実施例のコンパイラによれば、組み込み手続等の呼出はオブジェクト言語における手続呼出に置き換えられるだけなので、そのような問題は生じない。

上記図5～図9を参照しながら説明した実施例では、ソースプログラム中の変換すべき対象が互いに同一ではない場合を採り上げた。すなわち、図7Aに示した例では、2つの変換すべき対象は、呼び出すべき関数は互いに同じであったが、その引数は互いに異なっていた。

ところが、ソースプログラム中に複数の互いに同一の変換すべき対象が存在することがある。この場合、もし、図5に示したコンパイラ10を用いてそのソースプログラムをコンパイルすると、複数の互いに同一の手続コード（オンラインコード）を生成することになり、コンパイラの総出力コード量が必要以上に多くなってしまう。以下、この問題を解決したコンパイラについて説明する。

図10は、本発明の他の形態のコンパイラの機能ブロック図である。図10において使用する符号のうち、図5において先に使用しているものは同じ機能

を持つユニットを表す。すなわち、コンパイラ30において、検出部11、変換部12、展開部13は、基本的に図5に示したコンパイラ10のものと同一ユニットである。

コンパイラ30は、重複判定部31を備えることを特徴とする。重複判定部31は、検出部11によりソースプログラムにおいて検出された複数の変換すべき対象のうち、互いに同じものが存在するか否かを調べ、その結果を変換部12および展開部13に通知する。互いに同じ変換すべき対象が存在しなければ、変換部12および展開部13は、図5～図9を参照しながら説明した動作を実行する。一方、互いに同じ変換すべき対象が存在する場合は、変換部12は、それらをそれぞれ互いに同じ呼出に変換し、展開部13はそれらの呼出により共有的に呼び出される手続コード（オンラインコード）を1つだけ生成する。

図11は、図10に示したコンパイラの動作を説明するためのフローチャートである。

ステップS11では、まず、重複判定部31が使用するメモリ領域R（管理テーブル）をクリアする。なお、メモリ領域Rには、ソースプログラム中に存在する変換すべき対象の特徴が登録され、さらに、各特徴に対してユニークに割り当てられる手続名が登録される。ステップS12は、図6のステップS1と同様に、プログラムP内で変換すべき対象C1, ..., Cnを検出する。

ステップS13～S19の処理は、ステップS12において検出された各変換すべき対象C1, ..., Cnについて実行される。ステップS13では、図6のステップS2と同様に、変換すべき対象Ciの特徴Aiを抽出する。ステップS14では、特徴Aiが、先にステップS14～S19の処理が終了してい

る変換すべき対象  $C_1, \dots, C_{i-1}$  の特徴の中のいずれかと同じであるか否かを調べる。特徴  $A_i$  が、先の処理がなされた変換すべき対象  $C_1, \dots, C_{i-1}$  の特徴の中のいずれかと同じであればステップ S 19 へ進み、いずれにも一致しなければステップ S 15 へ進む。

- 5      ステップ S 15 では、メモリ領域 R をサーチし、変換すべき対象  $C_i$  に対してプログラム P 内でユニークな手続名  $f_i$  を生成する。ステップ S 16 では、図 6 のステップ S 4 と同様に、変換すべき対象  $C_i$  を手続  $f_i$  の呼出に置換する。ステップ S 17 では、図 6 のステップ S 5 と同様に、特徴  $A_i$  に対応する手続コード  $S_i$  を生成する。この手続コード  $S_i$  の手続名として  $f_i$  が指定される。さらに、ステップ S 18 において、変換すべき対象  $C_i$  の特徴  $A_i$  および対応する手続コード  $S_i$  を呼び出すための手続名としての  $f_i$  がメモリ領域 R に登録される。
- 10

- 一方、ステップ S 19 では、変換すべき対象  $C_i$  を、特徴  $A_j$  に対応する手続コード  $S_j$  を呼び出すための手続  $f_j$  の呼出に置換する。なお、ステップ S
- 15      19 が実行された場合は、新たな手続コードが生成されず、また、メモリ領域 R への新たな登録も行われなない。

- 上記ステップ S 13 ~ S 19 の処理を全ての変換すべき対象  $C_1, \dots, C_n$  について実行すると、プログラム P において各変換すべき対象  $C_1, \dots, C_n$  が手続  $f_1, \dots, f_m$  ( $m \leq n$ ) の呼出に置換され、また、各手続  $f_1, \dots, f_m$  の呼出にそれぞれ対応して手続コード  $S_1, \dots, S_m$  が生成される。そして、ステップ S 20 において、変換すべき対象を手続の呼出に置き換えることにより得られるプログラム P'、および生成された手続コードが出力される。なお、プログラム P' および手続コード  $S_1, \dots, S_m$  は、互いに同じファイ
- 20

ルに出力されてもよいし、互いに異なるファイルに出力されてもよい。

このように、コンパイラ 30 によれば、あるソースプログラム内に存在する互いに重複した変換すべき対象に対して、重複して手続コードを生成することなく、必要最小限の手続コードを生成するので、コンパイラの総出力コード量が減少する。

次に、具体的な実施例を説明する。以下では、コンパイラ 30 に対して図 12 A に示すソースプログラムが入力された場合を想定し、関数 SUM の呼出を変換対象として説明する。なお、コンパイラ 30 は、図 8 に示したテンプレートを利用するものとする。

10 ソースプログラムが入力されると、コンパイラ 30 は、そのソースプログラムをスキャンすることによって関数 SUM の呼出を検出する。図 12 A に示す例においては、関数 SUM は、4 行目の「SUM(A)」および「SUM(A2)」、5 行目の「SUM(N(51:100))」、および 6 行目の「SUM(M(51:200))」により呼ばれていることが検出される。

15 コンパイラ 30 は、まず、「SUM(A)」についてその特徴を抽出し、その特徴に従ってオンライン展開を実行する。このオンライン展開の結果は、上述したコンパイラ 10 による展開結果と同じであり、図 9 A に示した手続コードを生成する。また、「SUM(A)」が「SUM\_SAMPLE\_1(A)」に置き換えられる動作も上述した通りである。

20 ただし、コンパイラ 30 においては、同じ手続コードを重複して生成することを回避するために、順次、生成した手続コードを呼び出すための呼出の名前と、その手続の特徴が管理テーブルに登録される。なお、「特徴」は、引数の型、引数の次元数、引数の各次元の下限值および上限値である。したがって、

「SUM(A)」についてオンライン展開が実行されると、図13Aに示すように、  
「SUM(A)」に対応するレコードが登録される。

続いて、コンパイラ30は、「SUM(A2)」の特徴を抽出し、図13Bに示す  
ように、管理テーブルに既に登録されている呼出の特徴とその抽出した特徴と  
5 を比較する。ここでは、「SUM\_SAMPLE\_1(A)」および「SUM(A2)」の特徴が互  
いに完全に一致しているので、「SUM(A2)」に対しては新たに手続コードを生  
成することなく、「SUM(A)」に対して生成された手続コードを「SUM(A2)」か  
ら呼び出すようにするために、「SUM(A2)」を「SUM\_SAMPLE\_1(A2)」に変換す  
る。この結果、ソースプログラムの4行目は、以下のように変換されることに  
10 なる。

$$B = \text{SUM}(A) + \text{SUM}(A2) \rightarrow B = \text{SUM\_SAMPLE\_1}(A) + \text{SUM\_SAMPLE\_1}(A2)$$

したがって、この行が実行されると、「SUM\_SAMPLE\_1(A)」により図9Aに示  
した手続コードが呼び出され、さらに、「SUM\_SAMPLE\_1(A2)」によってもその  
同じ手続コードが呼び出されることになる。

15 なお、「SUM\_SAMPLE\_1(A)」および「SUM(A2)」の特徴が互いに完全に一致  
していたので、管理テーブルには新たなレコードは追加されず、管理テーブル  
は図13Aに示す状態を保持する。

続いて、コンパイラ30は、「SUM(N(51:100))」の特徴を抽出し、図13C  
に示すように、管理テーブルに既に登録されている呼出の特徴とその抽出した  
20 特徴とを比較する。ここでは、「SUM\_SAMPLE\_1(A)」及び「SUM(N(51:100))」  
の特徴が互いに一致していないので、「SUM(N(51:100))」に対応する手続コー  
ドを新たに生成する。この手続コードは、図9Bに示した手続コードと同じで  
ある。



そして、コンパイラ30は、管理テーブルをサーチし、先に登録されていない名前（ここでは、「SUM\_SAMPLE\_2」）を決定し、それを「SUM(N(51:100))」に割り当てる。この後、ソースプログラムの5行目が下記のように書き換えられる。この処理は、図5～図9を参照しながら説明した処理と同じである。

5       WRITE(\*,\*) SUM(N(51:100)) → WRITE(\*,\*) SUM\_SAMPLE\_2(N(51:100))

なお、新たに使用されることとなった名前および対応する特徴は、管理テーブルに登録される。

さらに、コンパイラ30は、「SUM(M(51:200))」に対して同様の処理を実行する。この場合、図13Dに示すように、「SUM(M(51:200))」の特徴は、先に  
10   登録されている「SUM\_SAMPLE\_2」の特徴と類似しているが、完全には一致していないので、「SUM(M(51:200))」に対応する手続コードを新たに生成する。この新たに生成される手続コードを図14に示す。

そして、コンパイラ30は、管理テーブルをサーチし、先に登録されていない1つの名前（「SUM\_SAMPLE\_3」）を決定し、それを「SUM(M(51:200))」に  
15   対して割り当てる。この後、ソースプログラムの6行目が下記のように書き換えられる。

      WRITE(\*,\*) SUM(M(51:200)) → WRITE(\*,\*) SUM\_SAMPLE\_3(M(51:200))

このように、コンパイラ30は、図12Aに示すソースプログラムが入力されると、図12Bに示すオブジェクトコード、および図9A、図9B、および  
20   図14に示す手続コードを出力する。ここで、図9Aに示す手続コードは、2つの呼出により共有的に呼び出される。すなわち、上記実施例では、ソースプログラムから変換すべき対象が4つ検出されたが、生成された手続コードは3つである。このように、ソースプログラムの中に複数の互いに同じ変換対象が

あった場合、コンパイラ 30 の総出力コード量は図 5 に示したコンパイラ 10 と比較して減少する。

5      なお、図 10 ～図 14 に示した実施例では、図 8 に示したテンプレートを用いて関数 SUM の手続コードを生成したが、テンプレートの表現の抽象度を高めると、生成される手続コードの数をさらに減少させることができる。

10      図 15 は、図 8 に示したテンプレートと比較して抽象度の高い表現で作成したテンプレートの例である。図 8 に示したテンプレートは、引数の型、引数の次元数、および引数の各次元の下限值および上限値をパラメータとして手続を展開するフォーマットであったが、図 15 に示すテンプレートは、引数の型および引数の次元数のみをパラメータとして手続を展開するフォーマットである。したがって、図 15 に示すテンプレートを用いた場合、引数の型および引数の次元数が互いに同じであれば、引数の各次元の下限值および上限値の不一致は無視され、配列の大きさに因らない手続コードが作成される。

15      図 16 は、図 15 に示すテンプレートを用いて変換すべき対象を展開する際に比較されるパラメータを示す図である。ここでは、図 12 A に示したソースプログラムから検出された 4 つの変換すべき対象について示している。

20      図 15 に示すテンプレートを用いる場合、比較すべきパラメータは、上述したように、変換すべき対象の「引数の型」および「引数の次元数」である。従って、「SUM(A)」および「SUM(A2)」に対して互いに同じ手続コードが生成されることになり、同様に、「SUM(N(51:100))」および「SUM(M(51:200))」に対して互いに同じ手続コードが生成されることになる。すなわち、コンパイラ 30 は、「SUM(A)」および「SUM(A2)」に対して 1 つの手続コードを生成し、同様に、「SUM(N(51:100))」および「SUM(M(51:200))」に対しても 1 つの手続コ

ードを生成する。このとき、「SUM(A)」および「SUM(A2)」は、互いに同じ手続呼出（ここでは、「SUM\_SAMPLE\_1」とする。）に変換される。同様に、「SUM(N(51:100))」および「SUM(M(51:200))」も互いに同じ手続呼出（ここでは、「SUM\_SAMPLE\_2」とする。）に変換される。

- 5 図17は、図15に示すテンプレートを用いた場合のコンパイラ30の出力を示す図である。コンパイラ30は、図12Aに示したソースプログラムが入力されると、図15に示したテンプレートを用いた場合、図17に示すオブジェクトコード、および2つの手続コード（オンラインコード）を出力する。ここで、手続コードAは、「SUM\_SAMPLE\_1」により呼び出され、また、手続コードBは、「SUM\_SAMPLE\_2」により呼び出される。
- 10

- なお、図17に示す例では省略されているが、実際には、オブジェクトコードにおいて手続コードに対するインタフェース宣言が記述される必要がある。例えば、Fortran90 プログラムとしてオブジェクトコードを生成する場合は、ソースプログラム中の変換すべき対象を新たな手続呼出に変換する際、コンパイラが、その新たな手続呼出に関するインタフェース宣言を生成する。このことは、他の実施例においても同様である。
- 15

- 上記図5および図10に示した実施例では、コンパイラへの入力が1つのプログラムであることを前提としていたが、実際には、複数のソースプログラムを含むファイルを一括してコンパイルしたい場合が多々ある。以下では、複数のソースプログラムを含むファイルを一括してコンパイルするコンパイラについて説明する。
- 20

図18は、本発明のさらに他の形態のコンパイラの機能ブロック図である。図18において使用する符号のうち、図5または図10において先に使用して

いるものは同じ機能のユニットを表す。

コンパイラ40は、基本的には、図10に示したコンパイラ30と同じである。ただし、コンパイラ30に設けられる重複判定部31は、各ソースプログラム中に重複する変換すべき対象が存在するか否かを調べるのに対し、コンパイラ40に設けられる重複判定部41は、入力ファイル毎に重複する変換すべき対象が存在するか否かを調べる。重複判定部41による判定結果は、変換部12および展開部13に通知される。この通知を受けた変換部12および展開部13の動作は、基本的に、図10を参照しながら説明した通りである。

図19は、図18に示したコンパイラの動作を説明するためのフローチャートである。ここでは、コンパイラ40に入力されるソースファイルをファイルF、コンパイラ40によりコンパイルされたオブジェクトファイルをファイルF'と呼ぶことにする。なお、以下では、ファイルFは、1以上のソースプログラムP1, ..., Ptを含むものとする。また、この処理は、1つのソースファイルが入力されるごとに実行される。

ステップS21～S30の各処理は、基本的に、図11に示したステップS11～S20とそれぞれ同じである。ただし、ステップS21～S30では、ファイル単位で処理が行われる。すなわち、ステップS22では、ファイルFに含まれている1以上のソースプログラムから変換すべき対象が検出される。また、ステップS25では、変換すべき対象に対してファイルF内でユニークな名前を生成する。

コンパイラ40は、ソースプログラムP1, ..., Ptを含むファイルFが入力されると、上記ステップS21～S30の処理を実行することにより、各変換すべき対象がそれぞれ手続の呼出に置き換えられたオブジェクトプログラム

P'1, ..., P'1 を生成し、また、各手続の呼出にそれぞれ対応して手続コード S1, ..., S<sub>m</sub> を生成する。そして、これらのオブジェクトプログラムおよび生成された手続コードは、同一のファイルに出力される。

- なお、手続コードは、生成される毎に、順次、1つずつ出力してもよいし、
- 5 プログラム単位の処理が終了する毎に出力してもよいし、あるいはファイル単位の処理が終了した時点で一括して出力してもよい。また、ソースコードが複数のファイルから構成される場合には、各ファイル毎に上記処理を繰り返し実行すればよい。

- 10      このように、コンパイラ 40 によれば、複数のソースプログラムを含むファイル内に存在する互いに重複した変換すべき対象に対して、重複して手続コードを生成することなく、必要最小限の手続コードを生成するので、コンパイラの総出力コード量がさらに減少する。

- 次に、具体的な実施例を説明する。以下では、コンパイラ 40 に対して図 20 に示すソースファイル (tiny.f) が入力された場合を想定し、関数 SUM の呼出を変換対象として説明する。なお、コンパイラ 40 は、図 8 に示したテンプレートを利用するものとする。

- ソースファイルが入力されると、コンパイラ 40 は、そのソースファイルに含まれている 2 つのプログラム（メインプログラムおよびサブプログラム）をスキャンすることにより関数 SUM の呼出を検出する。図 20 に示す例においては、関数 SUM は、メインプログラムの「SUM(A)」及び「SUM(N(51:100))」、およびサブプログラムの「SUM(Q)」により呼ばれていることが検出される。

続いて、コンパイラ 40 は、検出した各変換すべき対象の特徴（引数の型、引数の次元数など）に基づいて、生成すべき手続コードが同一になるものが存

在するか否かを調べる。ここでは、「SUM(A)」および「SUM(Q)」の特徴が互いに同じである。従って、「SUM(A)」および「SUM(Q)」に対しては、これらにより共有される1つの手続コードが生成される。なお、「SUM(N(51:100))」に対しては、1つの独立した手続コードが生成される。

- 5 図21は、図20に示すソースファイルが入力されたときのコンパイラ40の出力である。ここでは、呼出の名前は、手続の名前と、ファイル名と、そのファイルの中での登場順番との組合せにより決定されるものとしている。従って、「SUM(A)」および「SUM(Q)」は「SUM\_TINY\_1」に置き換えられている。また、「SUM(N(51:100))」は「SUM\_TINY\_2」に置き換えられている。そして、手続コードA及び手続コードBは、それぞれ「SUM\_TINY\_1」及び「SUM\_TINY\_2」により呼び出されることになる。

- このように、コンパイラ40によれば、互いに同じ変換すべき対象が同一ファイル内に存在する場合、それらにより共有される手続コードが生成される。このため、生成した手続コードをオブジェクトプログラムと同じファイルに出力するようにすれば、ファイルの取り扱いが簡単になる。例えば、入力ファイルのファイル名が「tiny.f」であるときは、出力ファイルのファイル名を「tiny.o」とする。そして、そのファイルにオブジェクトプログラムおよび手続コードの双方を出力する。

- 図22は、本発明のさらに他の形態のコンパイラの機能ブロック図である。
- 20 コンパイラ50は、複数のファイルが一括して入力されることを考慮し、入力された複数のファイルに存在する互いに同じ変換すべき対象を同一の手続呼出に変換すると共に、その手続呼出に対して1つの手続コードを生成する。この手続コードは、複数のファイルにより共通化され、複数の手続呼出により共有

的に呼び出される。

上記機能を実現するために、重複判定部51は、複数の入力ファイル中に重複する変換すべき対象が存在するか否かを調べ、その判定結果を変換部12および展開部13に通知する。この通知を受けた変換部12および展開部13の

5 動作は、基本的に、図10を参照しながら説明した通りである。

リンク編集部23は、既存の一般的なプログラムにより実現可能であり、コンパイル済みのコードとデータファイルとをリンクさせることにより実行可能プログラムを生成するプログラムである。1つのファイルは、一般に1つ以上のプログラム単位（サブルーチンや関数等）を含み、1つの実行可能ファイル  
10 は、1つ以上のファイルのソースコードをコンパイルし、それらをリンク編集して作成される。また、リンク編集部23は、ライブラリの生成機能も持っている。

図23は、図22に示したコンパイラの動作を説明するためのフローチャートである。ここでは、コンパイラ50に入力される複数のソースファイルをファイルF1, ..., Fs、コンパイラ50によりコンパイルされたオブジェクト  
15 ファイルをファイルF'1, ..., F'sと呼ぶことにする。また、各ソースファイルは、それぞれ1以上のソースプログラムを含むものとする。なお、図23に示す処理は、一連の複数のソースファイルが入力されるごとに実行される。

ステップS31～S40の各処理は、基本的に、図11に示したステップS  
20 11～S20とそれぞれ同じである。ただし、ステップS31～S40では、一連の複数のファイルを単位として処理が行われる。すなわち、ステップS32では、ファイルF1, ..., Fsに含まれている複数のソースプログラムから変換すべき対象が検出される。また、ステップS35では、変換すべき対象に

対して一連のファイル  $F_1, \dots, F_s$  内でユニークな名前を生成する。なお、出力ファイルをユーザが扱う通常のファイルとする場合には、オブジェクトコードと手続コード（オンラインコード）とが対応付けられる。

コンパイラ 50 は、ファイル  $F_1, \dots, F_s$  が入力されると、上記ステップ  
5 S 31 ~ S 40 の処理を実行することにより、各変換すべき対象がそれぞれ手続の呼出に置き換えられたオブジェクトプログラムを生成し、これらのオブジェクトプログラムを含むファイル  $F'_1, \dots, F'_s$  を出力する。また、コンパイラ 50 は、各手続の呼出にそれぞれ対応する手続コードを生成し、これらの手続コードを共通ファイル  $F_0$  に出力する。

10 なお、手続コードは、生成される毎に、順次、1 つずつ出力してもよいし、プログラム単位の処理が終了する毎あるいはファイル単位の処理が終了する毎に出力してもよい。また、手続コードは、オブジェクトファイルを含むファイルと同じファイルに出力してもよい。また、コンパイラ 50 は、出力ファイルを一時ファイルとして管理し、リンク編集処理をも行うことによって実行ファ  
15 イルを生成してもよい。

このように、コンパイラ 50 によれば、複数のソースファイル内に存在する互いに重複した変換すべき対象に対して、重複して手続コードを生成することなく、必要最小限の手続コードを生成するので、コンパイラの総出力コード量がさらに減少する。

20 次に、具体的な実施例を説明する。以下では、コンパイラ 50 に対して図 24 に示す 2 つのソースファイル（tiny1.f および tiny2.f）が入力された場合を想定し、関数 SUM の呼出を変換対象として説明する。なお、コンパイラ 50 は、図 8 に示したテンプレートを利用するものとする。



ソースファイルが入力されると、コンパイラ 50 は、ファイル `tiny1.f` に含まれているプログラム（メインプログラム）、およびファイル `tiny2.f` に含まれているプログラム（サブプログラム）をスキャンすることにより関数 `SUM` の呼出を検出する。図 24 に示す例においては、関数 `SUM` は、ファイル `tiny1.f`

5      の「SUM(A)」及び「SUM(N(51:100))」、およびファイル tiny2.f 「SUM(Q)」  
により呼ばれていることが検出される。

続いて、コンパイラ 40 は、検出した各変換すべき対象の特徴（引数の型、引数の次元数など）に基づいて、生成すべき手続コードが同一になるものが存在するか否かを調べる。ここでは、「SUM(A)」および「SUM(Q)」の特徴が互いに同じである。従って、「SUM(A)」および「SUM(Q)」に対しては、これらにより共有される 1 つの手続コードが生成される。なお、「SUM(N(51:100))」に対しては、1 つの独立した手続コードが生成される。

図25は、図24に示すソースファイルが入力されたときのコンパイラ50の出力である。ここでは、呼出の名前は、手続の名前と、一連の複数の入力ファイルの中での登場順番との組合せにより決定されるものとしている。したがって、「SUM(A)」及び「SUM(Q)」は「SUM\_1」に置き換えられている。また、「SUM(N(51:100))」は「SUM\_2」に置き換えられている。そして、手続コードAおよび手続コードBは、それぞれ「SUM\_1」及び「SUM\_2」により呼び出されることになる。

20 このように、コンパイラ 50 は、手続コードは同時にコンパイルされる一連のファイルにより共有される。なお、リンク編集処理では、オブジェクトファイルにおいて手続コードが格納されているファイルが指定される。このリンク処理は、コンパイラ 50 により実行されてもよいし、あるいは、図 22 に示し

た構成のように、コンパイラとは別のモジュールにより実行されてもよい。コンパイラによりリンク編集処理が実行される場合には、手続コードを一時ファイルに格納し、コンパイル後にその一時ファイルを削除するようにすれば、ソースとの対応付けにおいて問題が生じにくくなる。

- 5     上記実施例では、変換すべき対象として「関数 SUM」を採り上げたが、他の関数やサブルーチンなどであっても同様に処理される。以下では、図5に示したコンパイラ10に図26Aに示すソースプログラムが入力された場合を想定し、「A\*\*N」を変換すべき対象とする。「A\*\*N」は、「AのN乗」を表す。ここで、「A」は、実数型、複素数型、または整数型のスカラであり、
- 10    「N」は、整数型のスカラであるものとする。

コンパイラ10は、ソースプログラムから「A\*\*N」を検出すると、それを「POW(A, N)」に変換することによりオブジェクトプログラムを生成すると共に、「POW(A, N)」により呼び出されるべき手続コード（オンラインコード）を出力する。この手続の定義は、例えば、「Nが3以下のとき

15    に掛け算を実行し、Nが3よりも大きいときには巾乗演算を実行する。」である。

コンパイラ10は、「POW(A, N)」に対応する手続コードを生成する際、以下の3つの特徴を参照する。

- 特徴1：Aの型
- 20    特徴2：Nの値が得られるか否か
- 特徴3：Nの値が得られたとき、その値

コンパイラ10は、「POW(A, N)」に対応する手続コードを生成する際に利用するサンプルデータとして図27A～図27D、および図28Aおよ

び図28Bに示すテンプレートを用意する。これらのテンプレートでは、上記特徴1が抽象化されている。

図27Aに示すテンプレートは、「N」が「0」の場合に対応して記述されている。すなわち、「N=0」の場合は、「Aの0乗」は常に「1」なので、

5 テンプレートには予め「R=1」が記述されている。

図27B～図27Dに示す各テンプレートは、それぞれ「N」が「1」から「3」の場合に対応して記述されている。ここでは、「Nが小さいときには、巾乗演算を実行するよりも掛け算を実行した方が有利である」という前提に立ち、各テンプレートには、それぞれ掛け算式（「R=A」「R=A\*A」および「R=A\*A\*A」）が予め記述されている。

10

図28Aに示すテンプレートは、「N」が3よりも大きい場合に対応して記述されている。即ち、テンプレートには、巾乗演算式（「R=A\*\*A」）が予め記述されている。

図28Bに示すテンプレートは、ソースプログラムがコンパイルされる際に「N」の値が不明である場合に対応して記述されている。この場合、テンプレートには、「N」の値が決まった時点でその値に対応する演算式が選択されるような記述となっている。

15

コンパイラ10は、ソースプログラムが入力されると、各行をサーチすることにより「A\*\*N」という表現を検出する。図26Aに示すプログラムが入力された場合、4行目の「2\*\*LEN」および5行目の「(R\*2)\*\*2」が検出される。

20

まず、「2\*\*LEN」がオンライン展開される。「2\*\*LEN」は、プログラム SUBROUTINE の中で最初に登場した変換すべき対象なので、「POW\_SUBP\_

1) という名前が割り当てられる。続いて、上述した変換規則に従って以下の変換を行う。

$REAL::S(2**LEN-1) \rightarrow REAL::S(POW\_SUBP\_1(2, LEN)-1)$

そして、引数を解析することによって以下の情報を得る。

5      特徴1 : 型=INTEGER

特徴2 : Nの値は得られず

特徴3 : なし

コンパイラ10は、上記得られた情報により適切なテンプレートを選択してオンラインコードを生成する。すなわち、ここでは、「N」の値が得られていないので、図28Bに示したテンプレートが選択され、図26Bに示すオンラインコードAが生成される。

同様に、「 $(R*2)**2$ 」がオンライン展開される。「 $(R*2)**2$ 」は、プログラム SUBROUTINE の中で第2番目に登場した変換すべき対象なので、「POW\_SUBP\_2」という名前が割り当てられ、以下の変換が行われる。

15       $M=PAI*(R*2)**2 \rightarrow M=PAI*POW\_SUBP\_2((R*2), 2)$

そして、引数を解析することによって以下の情報を得る。

特徴1 : 型=REAL

特徴2 : Nの値は得られた

特徴3 :  $N=2$

20      コンパイラ10は、上記得られた情報に従って図27Cに示すテンプレートを選択し、図26Bに示すオンラインコードBを生成する。そして、コンパイラ10は、オブジェクトプログラムと共に、オンラインコードAおよびBを出力する。

なお、プログラムの実行時にオンラインコードを呼び出すためのオーバーヘッドを減少させるためには、本実施例のコンパレータで生成したオンラインコードを後続のコンパイラ（オブジェクト言語が機械後又はアセンブラ等であるコンパイラ）によりインライン展開してもよい。インライン展開は、上述した  
5 ように、プリプロセッサ方式のコンパイラで高級言語を対象とすると問題が生じることが多いが、オブジェクト言語が機械後又はアセンブラ等である通常のコンパイラでは、問題なくインライン展開を実行できることが多い。

上記コンパイラの機能は、コンピュータを用いて図6、図11、図19、または図23のフローチャートに示した処理を記述したプログラムを実行することにより実現される。そのプログラムを実行するコンピュータ60のブロック  
10 図を図29に示す。

CPU61は、図6、図11、図19、または図23のフローチャートに示した処理を記述したプログラムを記憶装置62からメモリ63にロードして実行する。記憶装置62は、例えばハードディスクであり、上記プログラム、および各種テンプレートを格納する。一方、メモリ63は、例えば半導体メモリ  
15 であり、CPU61の作業領域として使用される。図13Aなどに示した管理テーブルは、このメモリ63に作成される。

記憶媒体ドライバ64は、CPU61の指示に従って可搬性記憶媒体65にアクセスする。可搬性記憶媒体65は、半導体デバイス（ICカード等）、磁  
20 氣的作用により情報が入出力される媒体（フロッピーディスク、磁気テープなど）、光学的作用により情報が入出力される媒体（光ディスクなど）を含む。通信制御装置66は、CPU61の指示に従って網との間でデータを送受信する。

図30は、本発明に係わるソフトウェアプログラムなど（テンプレートを含む）の提供方法を説明する図である。本発明に係わるプログラムは、例えば、以下の3つの方法の中の任意の方法により提供される。

(a) コンピュータ60にインストールされて提供される。この場合、プログラム等は、たとえば、出荷前にプレインストールされる。

(b) 可搬性記憶媒体に格納されて提供される。この場合、可搬性記憶媒体65に格納されているプログラム等は、基本的に、記憶媒体ドライバ64を介して記憶装置62にインストールされる。

(c) 網上のサーバから提供される。この場合、基本的には、コンピュータ60がサーバに格納されているプログラム等をダウンロードすることによってそのプログラム等を取得する。

このように、本実施例のコンパイラは、プリプロセッサ方式であっても、ソースプログラム中における変換すべき対象の出現位置に係わらず、その変換すべき対象を容易に展開することができる。

また、手続呼出をインライン展開できる場合であっても、総出力コード量、およびコンパイル時間の点で本実施例のコンパイラの方が有利である。

#### (a) 出力コードの総量

図10、図18、および図22に示したコンパイラによれば、互いに同種の変換すべき対象が複数存在する場合には、それらに対して1つのオンラインコードが生成され、複数の手続呼出によりその生成されたオンラインコードが共有的に呼び出されるので、オブジェクトの総量が小さくなる。なお、図5に示したコンパイラの出力の総量は、インライン展開の場合のそれと比較して同程度である。

(b) コンパイル時間

一般に、コンパイラ内での最適化処理に要する時間は、コンパイル単位（1つの関数やサブルーチン等）の大きさの2乗～3乗に比例することが多い。このため、コンパイル単位のコード量が数倍になると、最適化のための処理時間は数十倍になることがある。

図5に示したコンパイラによれば、上述したように、出力コードの総量はインライン展開の場合と同程度だが、展開コード（オンラインコード）は独立したコンパイル単位として分割されて生成されるため、全体のコンパイル時間が短縮される。

また、図10、図18、および図22に示したコンパイラの場合は、生成されるオンラインコードの数が減少するので、コンパイル時間はさらに短縮される。

さらに、本実施例のコンパイラを使用した場合、オンライン展開後、入力コードのコンパイルをオンラインコードのコンパイルよりも先に実行するようにすれば、入力コードのエラーを速く検出することができる。インライン展開を使用した場合には、入力コードが大きくなるので、エラー検出に長い時間を要することが予想される。

産業上の利用可能性

本発明は、プリプロセッサ方式のコンパイラとして広く利用され得る。

# 請求の範囲

1. 入力された第1のプログラムをコンパイルして第2のプログラムを出力するコンパイラ装置であって、

5 上記第1のプログラムの一部を手続呼出に変換して上記第2のプログラムを作成する変換手段と、

上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成する生成手段と、

10 上記第2のプログラムおよび上記生成手段により生成されたコードを出力する出力手段と、

を有するコンパイラ装置。

2. 上記第1のプログラムから予め決められた特定のパターンを検出する検出手段をさらに有し、

15 上記変換手段は、その検出手段により検出されたパターンをそのパターンに対応する手続呼出に変換する請求項1に記載のコンパイラ装置。

3. 上記検出手段は、上記第1のプログラムから組込み手続を呼び出すための呼出を検出する請求項2に記載のコンパイラ装置。

4. 上記検出手段が上記第1のプログラムから複数の互いに同じパターンを検出した際には、上記変換手段は、それら複数のパターンを同一の手続呼出に変換する請求項1に記載のコンパイラ装置。

5. 1以上のプログラムを含む第1のファイルが入力され、その1以上のプログラムをコンパイルし、そのコンパイルされた1以上のプログラムを含む第2のファイルを出力するコンパイラ装置であって、



上記第1のファイルに含まれている1以上のプログラムから予め決められた特定のパターンを検出する検出手段と、

上記検出手段により検出されたパターンをそのパターンに対応する手続呼出に変換することにより上記第1のファイルに含まれている1以上のプログラム

5 をコンパイルする変換手段と、

上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成する生成手段と、

上記変換手段によりコンパイルされたプログラムおよび上記生成手段により生成されたコードを出力する出力手段とを有し、

10 上記変換手段が上記第1のファイルから複数の互いに同じパターンを検出し  
た際には、上記変換手段は、それら複数のパターンを同一の手続呼出に変換するコンパイラ装置。

6. 上記出力手段は、上記生成手段により生成されたコードを上記第2のファイルに出力する請求項5に記載のコンパイラ装置。

15 7. 各ファイルがそれぞれ1以上のプログラムを含む複数のファイルが入力され、それら入力ファイルに含まれているプログラムをコンパイルするコンパイラ装置であって、

上記入力ファイルに含まれているプログラムから予め決められた特定のパターンを検出する検出手段と、

20 上記検出手段により検出されたパターンをそのパターンに対応する手続呼出に変換することにより上記入力ファイルに含まれているプログラムをコンパイルする変換手段と、

上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成す

る生成手段と、

上記変換手段によりコンパイルされたプログラムおよび上記生成手段により生成されたコードを出力する出力手段とを有し、

上記変換手段が上記入力された複数のファイルから複数の互いに同じパターン

5 ンを検出した際には、上記変換手段は、それら複数のパターンを同一の手続呼出に変換するコンパイラ装置。

⑧  
claim 1  
means 形式

⑧. 入力された第1のプログラムをコンパイルして第2のプログラムを出力するコンパイル方法であって、

10 上記第1のプログラムの一部を手続呼出に変換して上記第2のプログラムを作成するステップと、

上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成するステップと、

上記第2のプログラムおよび上記生成手段により生成されたコードを出力するステップと、

15 を有するコンパイル方法。

⑨. コンピュータに、入力された第1のプログラムをコンパイルして第2のプログラムを出力させる、ためのプログラムを格納する記憶媒体であって、

上記第1のプログラムの一部を手続呼出に変換して上記第2のプログラムを作成させ、

20 上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成させ、

上記第2のプログラムおよび上記生成手段により生成されたコードを出力させるプログラムを格納する記憶媒体。

40

# 要 約 書

コンパイラ（10）は、検出部（11）、変換部（12）、展開部（13）を有する。検出部（11）は、入力されたソースプログラムから予め決められた対象を検出する。変換部（12）は、検出部（11）によって検出された対象を手続呼出に変換する。展開部（13）は、検出部（12）により得られた手続呼出によって呼び出されるべき手続の定義を記述したオンラインコードを生成する。コンパイラ（10）は、検出部（11）によって検出された対象が呼出手続に置き換えられたプログラム、およびその呼出手続に対応するオンラインコードを出力する。

1/30

```

-----
1      SUBROUTINE SUB(A,N)
2      INTEGER N
3      REAL A(ABS(N))
4      WRITE(*,*) A
5      END SUBROUTINE
-----

```

図 1 A

```

-----
1      SUBROUTINE SUB(A,N)
2      INTEGER N
          IF (N.GE.0) THEN      ! 展開コード
              TMP = N           ! 展開コード
          ELSE                   ! 展開コード
              TMP = -N          ! 展開コード
          END IF                ! 展開コード
3      REAL A(TMP)
4      WRITE(*,*) A
5      END SUBROUTINE
-----

```

図 1 B

2/30

```
-----
1      char *copy_string(char *s)
2      {
3          int i;
4          char *buffer = (char*)malloc(strlen(s) + 1);
5
6          for (i = 0; s[i] != '\0'; ++i)
7              buffer[i] = s[i];
8
9          return buffer;
10     }
```

2A

```
-----
1      char *copy_string(char *s)
2      {
3          int i;
4          char *p; /* 展開コード */
5          int tmp; /* 展開コード */
6          tmp = 0; /* 展開コード */
7          for (p = s; *p != '\0'; ++p) /* 展開コード */
8              ++tmp; /* 展開コード */
9          char *buffer = (char*)malloc(tmp + 1);
10         for (i = 0; s[i] != '\0'; ++i)
11             buffer[i] = s[i];
12
13         return buffer;
14     }
```

2B

```

-----
1      IF (Z.GT.EPS) THEN
2          A=B1
3      ELSE IF (ABS(Z).LE.EPS) THEN
4          A=B2
5      ELSE
6          A=B3
7      END IF
-----

```

3A

```

-----
1      IF (Z.GT.EPS) THEN
2          A=B1
3a     ELSE
        IF (Z.GE.0.0) THEN      ! 展開コード
            TMP = Z             ! 展開コード
        ELSE                     ! 展開コード
            TMP = -Z            ! 展開コード
        END IF                  ! 展開コード
3b     IF (TMP.LE.EPS) THEN
4         A=B2
5     ELSE
6         A=B3
3c     END IF
7     END IF
-----

```

3B

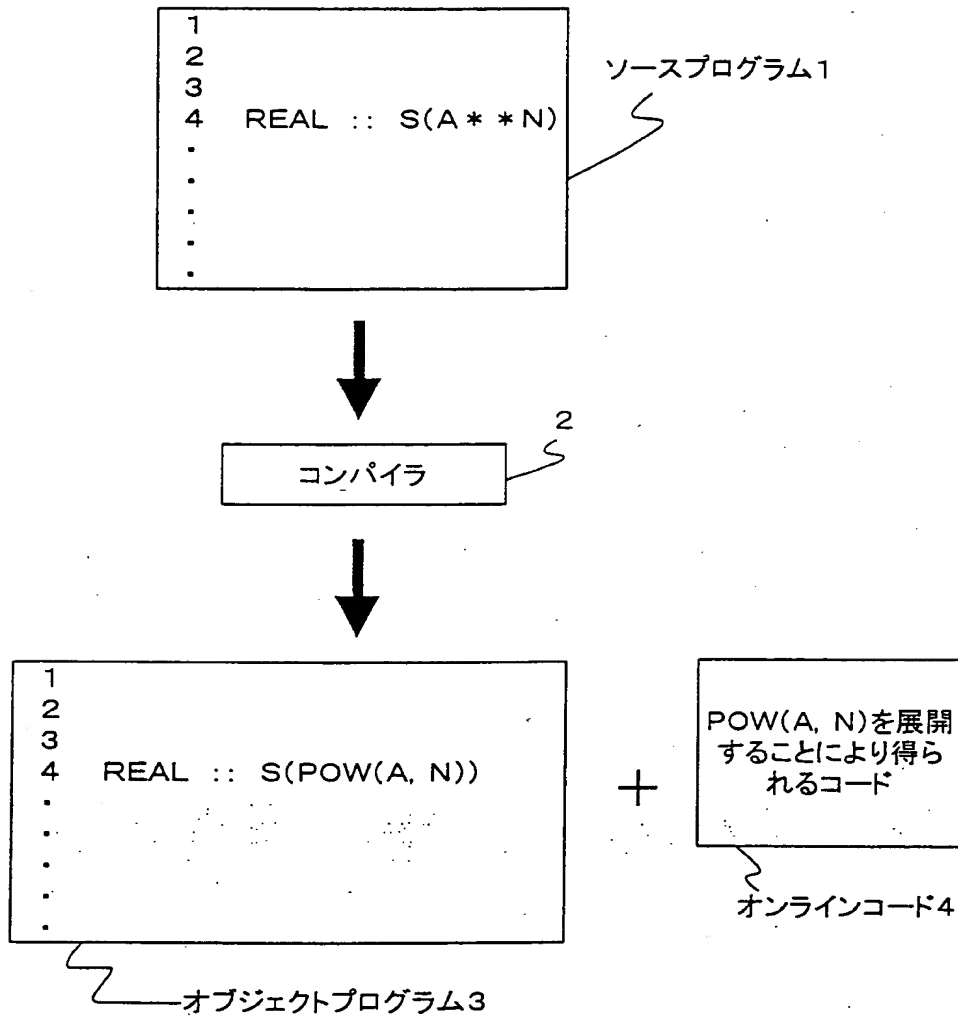


図4

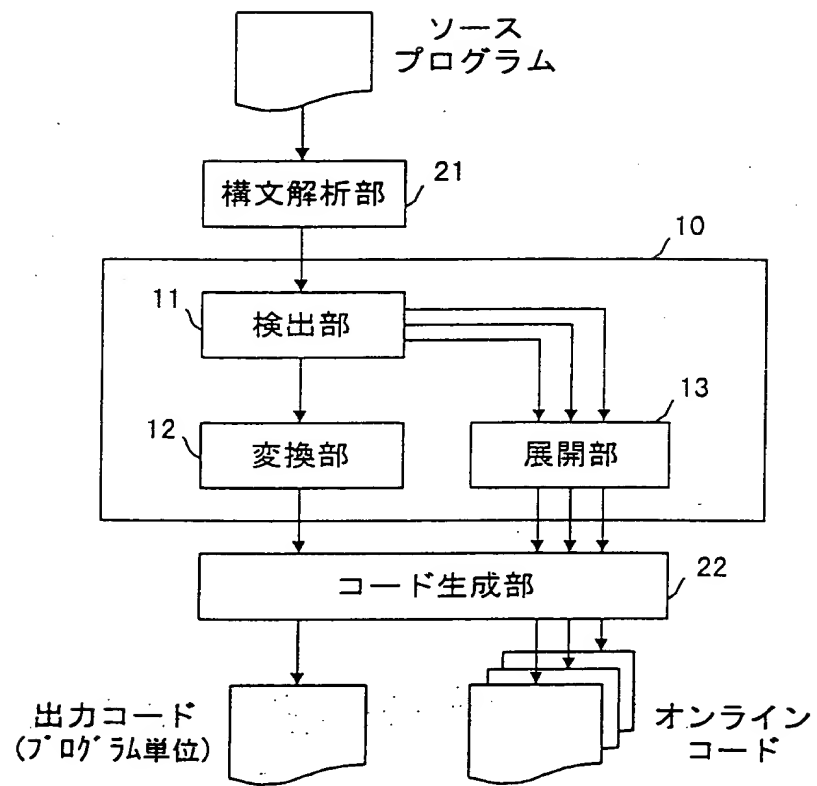


図 5



入力：プログラム単位 P  
出力：P を修正した P' と、手続 S<sub>1</sub>, ..., S<sub>n</sub> (0 ≤ n)

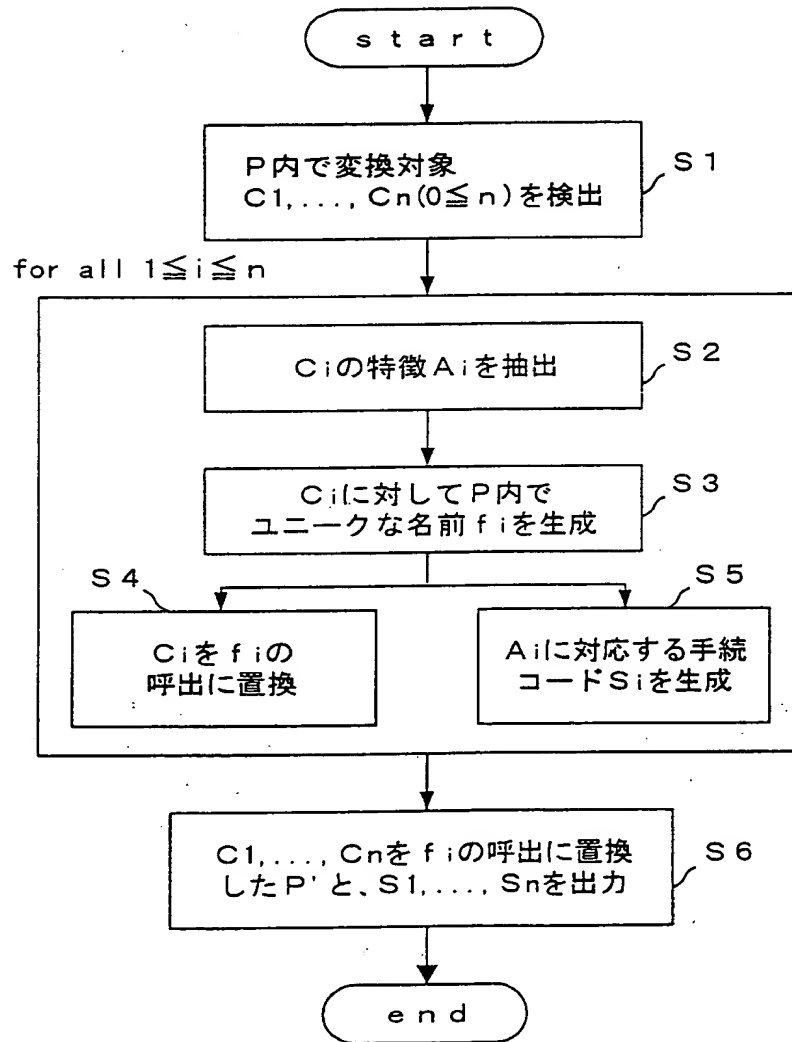


図 6

7/30

```

-----
1      PROGRAM SAMPL
2      INTEGER N(100)
3      REAL A(10,20,30),B
4      ...
5      B = SUM(A)
6      WRITE(*,*) SUM(N(51:100))
7      END
-----

```

7 A

```

-----
1      PROGRAM SAMPL
2      INTEGER N(100)
3      REAL A(10,20,30),B
4      ...
5      B = SUM_SAMPL_1(A)
6      WRITE(*,*) SUM_SAMPL_2(N(51:100))
7      END
-----

```

7 B

8/30

---

```

arg-type FUNCTION SUM(X)
arg-type X(lb(1):ub(1), ..., lb(m):ub(m))
SUM = 0
DO 999 Im = lb(m), ub(m)
    :
DO 999 I1 = lb(1), ub(1)
    SUM = SUM+X(I1,...,Im)
999 CONTINUE
RETURN
END

```

---

9/30

```
-----
REAL FUNCTION SUM_SAMPL_1(X)
REAL X(1:10,1:20,1:30)
SUM_SAMPL_1 = 0
DO 999 I3 = 1, 30
DO 999 I2 = 1, 20
DO 999 I1 = 1, 10
    SUM_SAMPL_1 = SUM_SAMPL_1+X(I1,I2,I3)
999 CONTINUE
RETURN
END
-----
```

9 A

```
-----
INTEGER FUNCTION SUM_SAMPL_2(X)
INTEGER X(51:100)
SUM_SAMPL_2 = 0
DO 999 I1 = 51, 100
    SUM_SAMPL_2 = SUM_SAMPL_2+X(I1)
999 CONTINUE
RETURN
END
-----
```

9 B

10/30

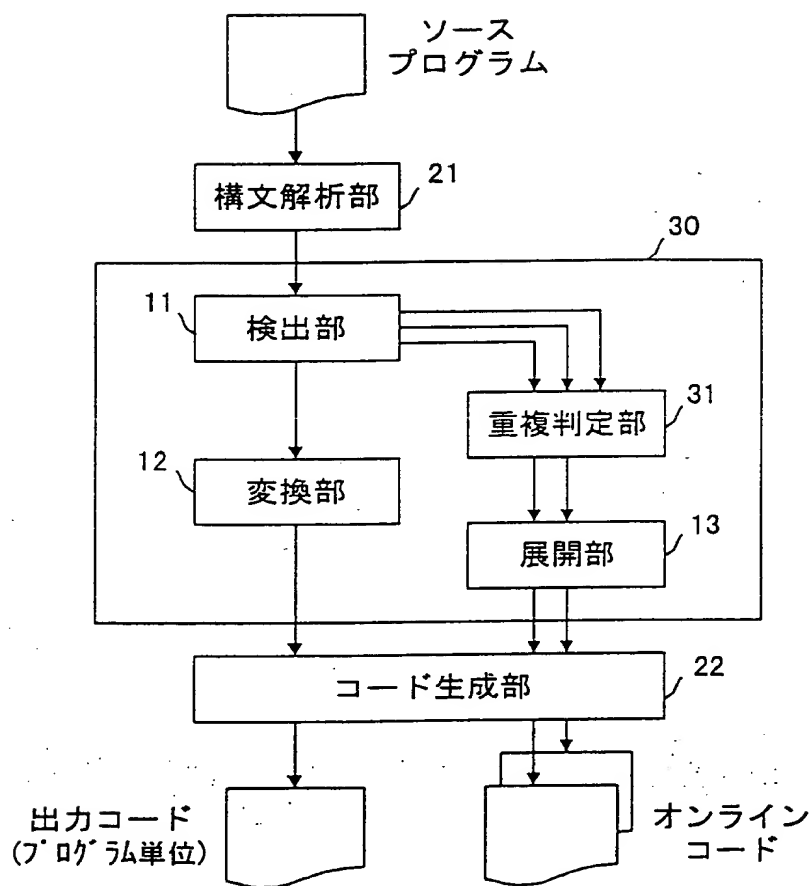


図 10

11/30

入力：プログラム単位 P  
出力：P を修正した P' と、手続 S<sub>1</sub>, ..., S<sub>m</sub> (0 ≤ m ≤ n)

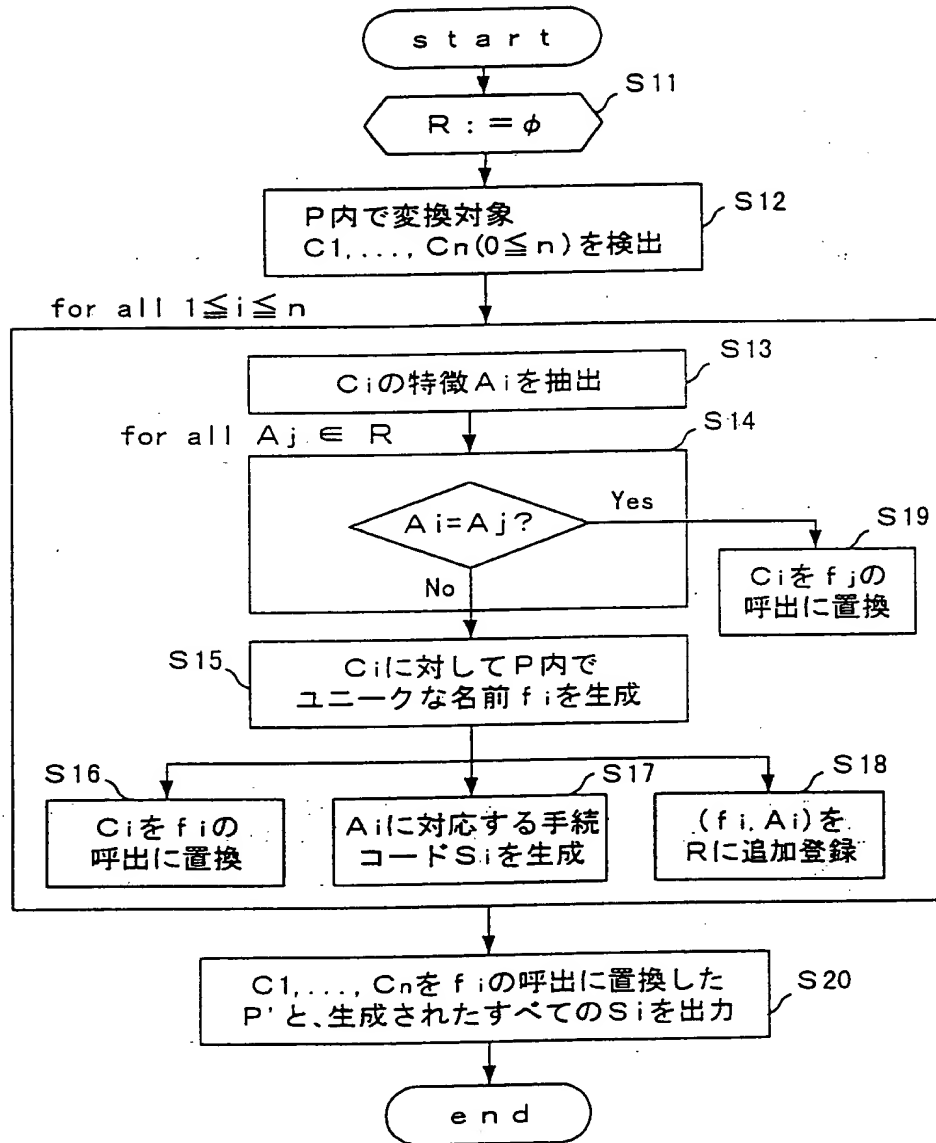


図 1 1

12/30

```

-----
1      PROGRAM SAMPL
2      INTEGER N(100),M(200)
3      REAL A(10,20,30),A2(10,20,30),B
4      ...
5      B = SUM(A)+SUM(A2)
6      WRITE(*,*) SUM(N(51:100))
7      WRITE(*,*) SUM(M(51:200))
8      END
-----

```

12 A

```

-----
1      PROGRAM SAMPL
2      INTEGER N(100),M(200)
3      REAL A(10,20,30),A2(10,20,30),B
4      ...
5      B = SUM_SAMPL_1(A)+SUM_SAMPL_1(A2)
6      WRITE(*,*) SUM_SAMPL_2(N(51:100))
7      WRITE(*,*) SUM_SAMPL_3(M(51:200))
8      END
-----

```

12 B

呼出し	arg-type	m	lb(1)	ub(1)	lb(2)	ub(2)	lb(3)	ub(3)
SUM_SAMPL__1	REAL	3	1	10	1	20	1	30

図13A

呼出し	arg-type	m	lb(1)	ub(1)	lb(2)	ub(2)	lb(3)	ub(3)
SUM_SAMPL__1	REAL	3	1	10	1	20	1	30
当該呼出	REAL	3	1	10	1	20	1	30

図13B

呼出し	arg-type	m	lb(1)	ub(1)	lb(2)	ub(2)	lb(3)	ub(3)
SUM_SAMPL__1	REAL	3	1	10	1	20	1	30
当該呼出	INTEGER	1	51	100	—	—	—	—

図13C

呼出し	arg-type	m	lb(1)	ub(1)	lb(2)	ub(2)	lb(3)	ub(3)
SUM_SAMPL__1	REAL	3	1	10	1	20	1	30
SUM_SAMPL__2	INTEGER	1	51	100	—	—	—	—
当該呼出	INTEGER	1	51	200	—	—	—	—

図13D



~~14/30~~

```

      INTEGER FUNCTION SUM_SAMPL_3(X)
      INTEGER X(51:200)
      SUM_SAMPL_3 = 0
      DO 999 I1 = 51, 200
         SUM_SAMPL_3 = SUM_SAMPL_3+X(I1)
999  CONTINUE
      RETURN
      END

```

15/30

---

```

arg-type FUNCTION SUM(X)
arg-type X(⋮, ⋮, ⋮, ⋮)
           m個 ← 抽象化
SUM = 0
DO 999 Im = LBOUND(X,m), UBOUND(X,m)
  ⋮
DO 999 I1 = LBOUND(X,1), UBOUND(X,1)
  SUM = SUM+X(I1,⋮,Im)
999 CONTINUE
RETURN
END

```

---

呼出し	<i>arg-type</i>	<i>m</i>
SUM(A)	REAL	3
SUM(A2)	REAL	3
SUM(N(51:100))	INTEGER	1
SUM(M(51:200))	INTEGER	1

図 16

17/30

```

PROGRAM SAMPL
INTEGER N(100),M(200)
REAL A(10,20,30),A2(10,20,30),B
...
B = SUM_SAMPL_1(A)+SUM_SAMPL_1(A2)
WRITE(*,*) SUM_SAMPL_2(N(51:100))
WRITE(*,*) SUM_SAMPL_2(M(51:200))
END
    
```

} オブジェクト  
コード

```

REAL FUNCTION SUM_SAMPL_1(X)
REAL X(:,:,:)
SUM_SAMPL_1 = 0
DO 999 I3 = LBOUND(X,3),UBOUND(X,3)
DO 999 I2 = LBOUND(X,2),UBOUND(X,2)
DO 999 I1 = LBOUND(X,1),UBOUND(X,1)
    SUM_SAMPL_1 = SUM_SAMPL_1+X(I1,I2,I3)
999 CONTINUE
RETURN
END
    
```

} 手続コード A

```

INTEGER FUNCTION SUM_SAMPL_2(X)
INTEGER X(:)
SUM_SAMPL_2 = 0
DO 999 I1 = LBOUND(X,1),UBOUND(X,1)
    SUM_SAMPL_2 = SUM_SAMPL_2+X(I1)
999 CONTINUE
RETURN
END
    
```

} 手続コード B

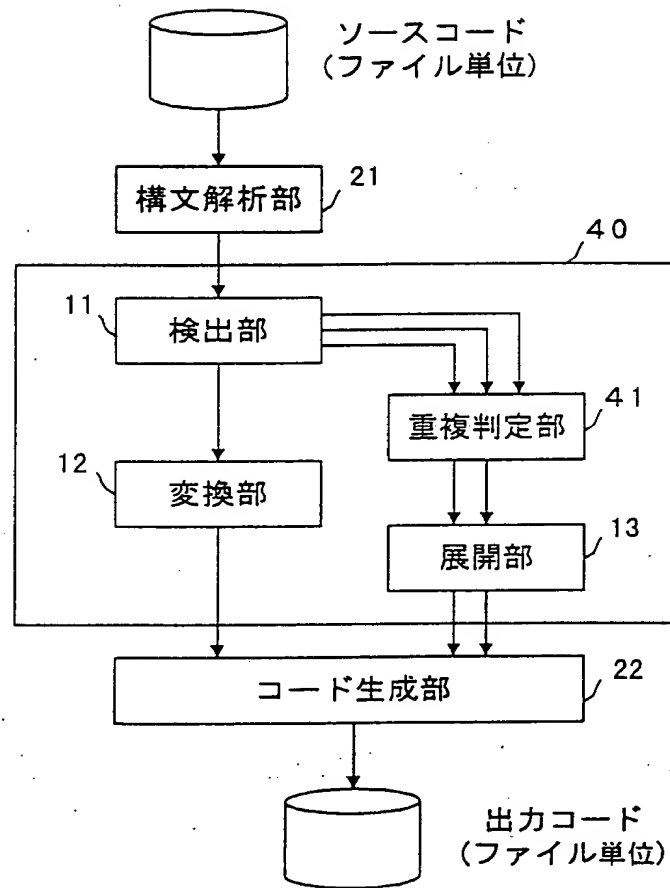


図 18

クレーム3

入力：プログラム単位  $P_1, \dots, P_t (1 \leq t)$  を含むファイル  $F$

出力：  $P_1, \dots, P_t$  をそれぞれ修正した  $P'_1, \dots, P'_t$  と

手続  $S_1, \dots, S_m (0 \leq m \leq n)$  を含むファイル  $F'$

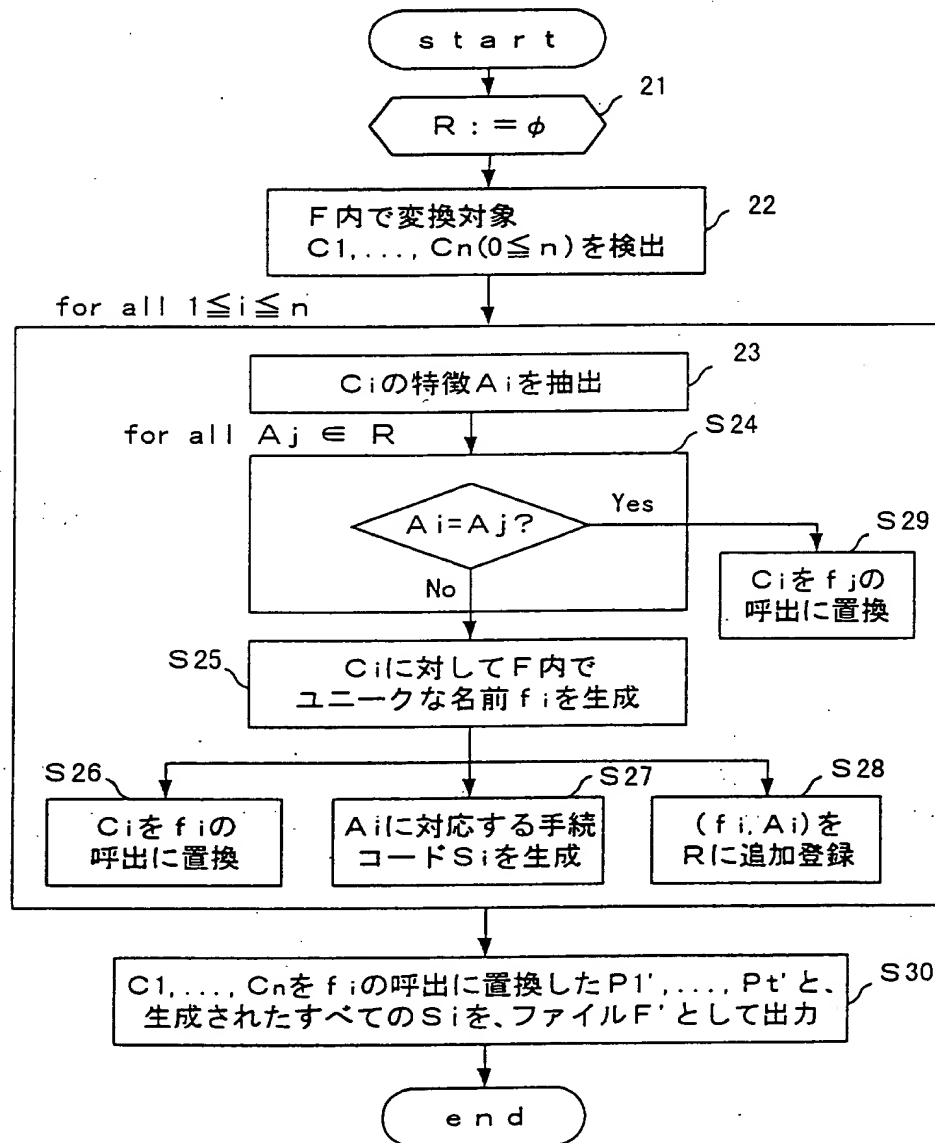


図 19

20/  
30

Ref. J99-00881

```
-----  
C-- main program ----  
PROGRAM SAMPL  
  INTEGER N(100)  
  REAL A(10,20,30), A2(10,20,30), B  
  ...  
  B = SUM(A)  
  B = SUM_AND_ADD(A,B)  
  WRITE(*,*) SUM(N(51:100))  
  END  
C-- subprogram ----  
  REAL FUNCTION SUM_AND_ADD(Q,S)  
  REAL Q(10,20,30), S  
  SUM_AND_ADD = SUM(Q)+S  
  RETURN  
  END  
C-- end of user programs ----  
-----
```

21/30

```

-----
C-- main program ----
  PROGRAM SAMPL
  INTEGER N(100)
  REAL A(10,20,30),A2(10,20,30),B
  ...
  B = SUM_TINY_1(A)
  B = SUM_AND_ADD(A,B)
  WRITE(*,*) SUM_TINY_2(N(51:100))
  END
C-- subprogram ----
  REAL FUNCTION SUM_AND_ADD(Q,S)
  REAL Q(10,20,30),S
  SUM_AND_ADD = SUM_TINY_1(Q)+S
  RETURN
  END
C-- end of user programs ----

  REAL FUNCTION SUM_TINY_1(X)
  REAL X(1:10,1:20,1:30)
  SUM_TINY_1 = 0
  DO 999 I3 = 1, 30
  DO 999 I2 = 1, 20
  DO 999 I1 = 1, 10
    SUM_TINY_1 = SUM_TINY_1+X(I1,I2,I3)
999 CONTINUE
  RETURN
  END

  INTEGER FUNCTION SUM_TINY_2(X)
  INTEGER X(51:100)
  SUM_TINY_2 = 0
  DO 999 I1 = 51, 100
    SUM_TINY_2 = SUM_TINY_2+X(I1)
999 CONTINUE
  RETURN
  END
-----

```

} 手続コード A

} 手続コード B



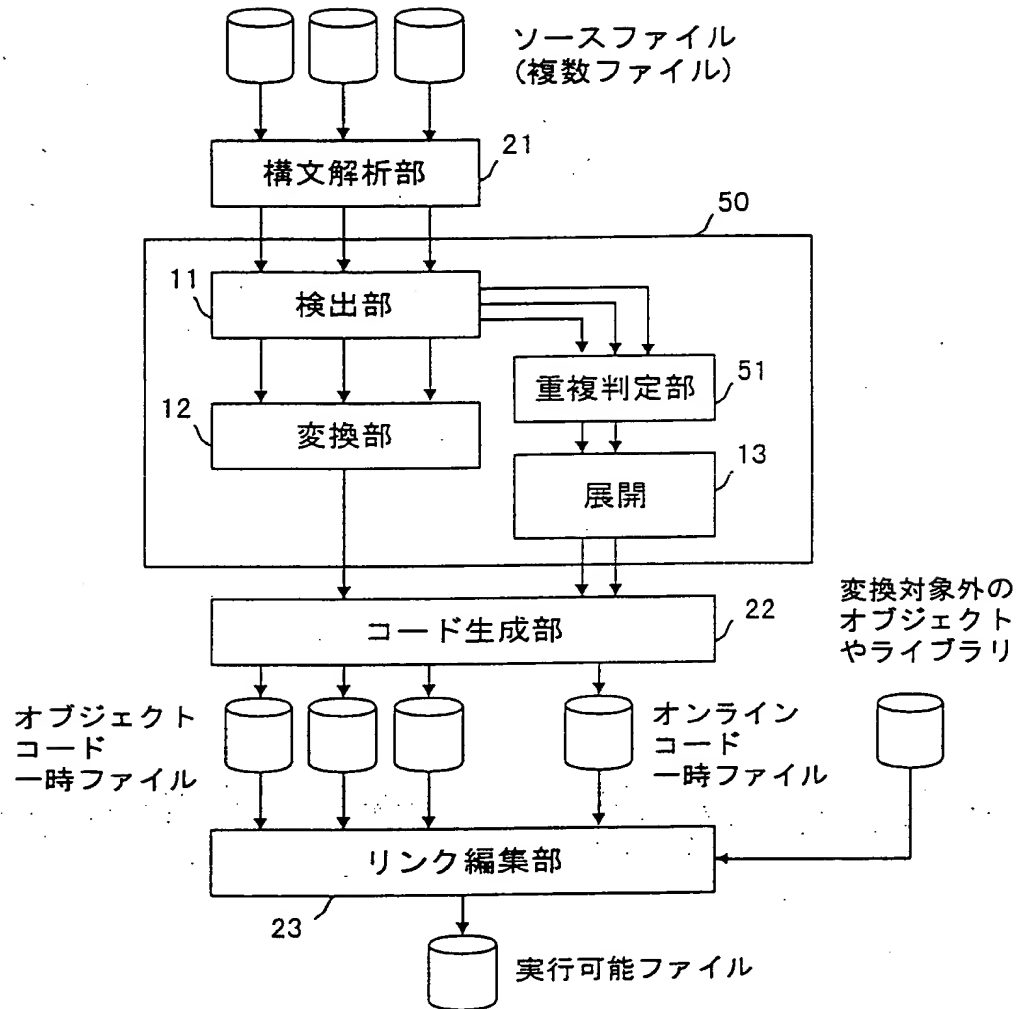


図 2 2

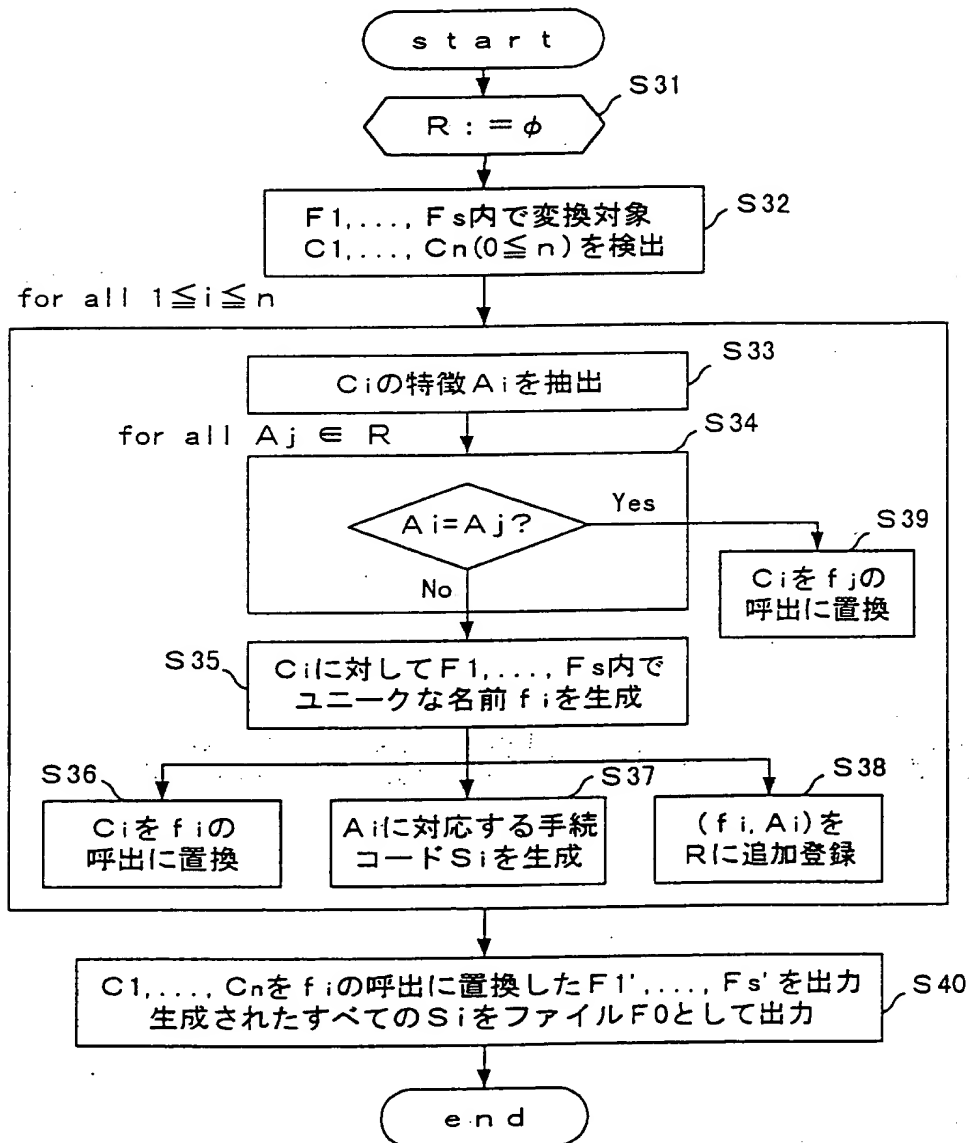
23/  
30クレーム 4入力：プログラム単位  $P_1, \dots, P_t$  ( $1 \leq t$ )  
を含むファイル  $F_1, \dots, F_s$  ( $1 \leq s$ )出力：  $F_1, \dots, F_s$  をそれぞれ修正した  $F_1', \dots, F_s'$  と  
手続  $S_1, \dots, S_m$  ( $0 \leq m \leq n$ ) を含むファイル  $F_0$ 

図 23

24/  
30

ファイル tiny1.f:

```
-----
C-- main program ----
  PROGRAM SAMPL
    INTEGER N(100)
    REAL A(10,20,30),A2(10,20,30),B
    ...
    B = SUM(A)
    B = SUM_AND_ADD(A,B)
    WRITE(*,*) SUM(N(51:100))
    END
C-- end of main program ----
-----
```

ファイル tiny2.f:

```
-----
C-- subprogram ----
  REAL FUNCTION SUM_AND_ADD(Q,S)
    REAL Q(10,20,30),S
    SUM_AND_ADD = SUM(Q)+S
    RETURN
    END
C-- end of subprogram ----
-----
```

```
C-- main program ----
```

C-- end of main program ----

C-- subprogram ----

C-- end of subprogram ----

手続コード A

手続コード B

26/30

```

1  SUBROUTINE SUBP(LEN)
2  REAL,PARAMETER :: PAI=3.14159, R=100.0
3  INTEGER LEN,M
4  REAL :: S(2**LEN-1)
5  ...
6  M=PAI*(R*2)**2
7  ...
8  END SUBROUTINE

```

26A

```

SUBROUTINE SUBP(LEN)
REAL,PARAMETER :: PAI=3.14159, R=100.0
INTEGER LEN,M
REAL :: S(POW_SUBP_1(2,LEN)-1)
...
M=PAI*POW_SUBP_2((R*2),2)
...
END SUBROUTINE

```

オブジェクト  
プログラム

```

FUNCTION POW_SUBP_1(A,N) RESULT(R)
INTEGER A,R
INTEGER N

SELECT CASE (N)
CASE (0)
R=1
CASE (1)
R=A
CASE (2)
R=A*A
CASE (3)
R=A*A*A
CASE DEFAULT
R=A**N
END SELECT
RETURN
END FUNCTION

```

オンライン  
コード A

```

FUNCTION POW_SUBP_2(A,N) RESULT(R)
REAL A,R
INTEGER N

R=A*A
RETURN
END FUNCTION

```

オンライン  
コード B

26B

27/  
30

27A

```
-----
FUNCTION name(A,N) RESULT(R)
  arg-type A,R
  INTEGER N

  R=1
  RETURN
END FUNCTION
-----
```

27B

```
-----
FUNCTION name(A,N) RESULT(R)
  arg-type A,R
  INTEGER N

  R=A
  RETURN
END FUNCTION
-----
```

27C

```
-----
FUNCTION name(A,N) RESULT(R)
  arg-type A,R
  INTEGER N

  R=A*A
  RETURN
END FUNCTION
-----
```

27D

```
-----
FUNCTION name(A,N) RESULT(R)
  arg-type A,R
  INTEGER N

  R=A*A*A
  RETURN
END FUNCTION
-----
```

28/30

---

```

FUNCTION name(A,N) RESULT(R)
  arg-type A,R
  INTEGER N

  R=A**N
  RETURN
END FUNCTION

```

---

28 A

---

```

FUNCTION name(A,N) RESULT(R)
  arg-type A,R
  INTEGER N

  SELECT CASE (N)
    CASE (0)
      R=1
    CASE (1)
      R=A
    CASE (2)
      R=A*A
    CASE (3)
      R=A*A*A
    CASE DEFAULT
      R=A**N
  END SELECT
  RETURN
END FUNCTION

```

---

28 B

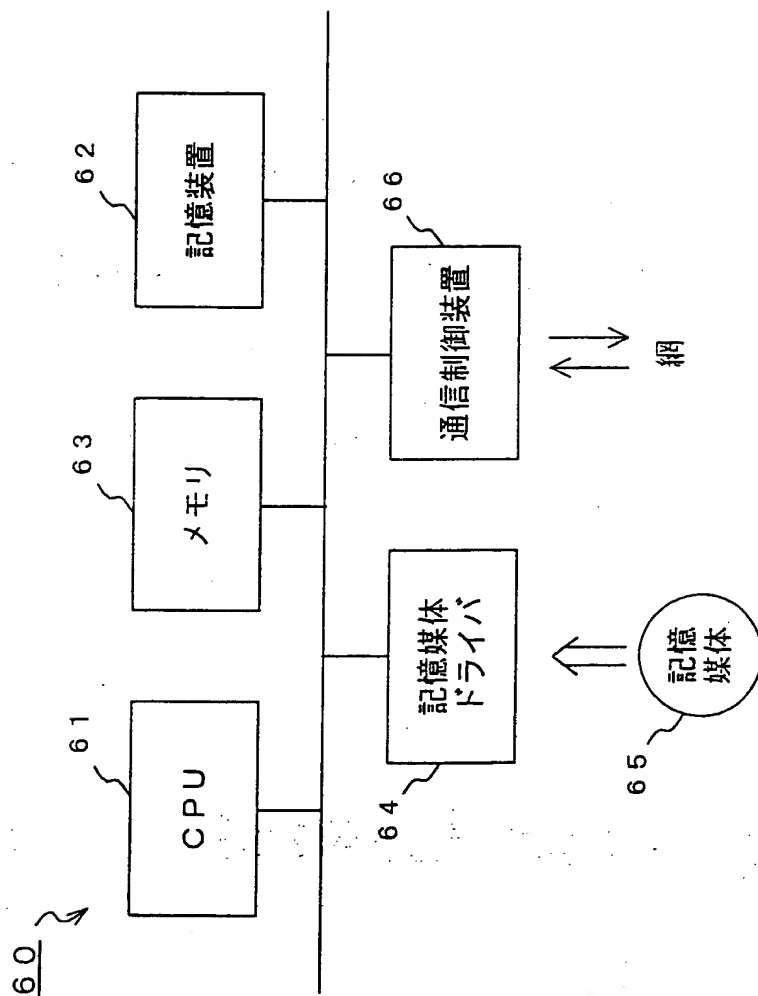


図 29



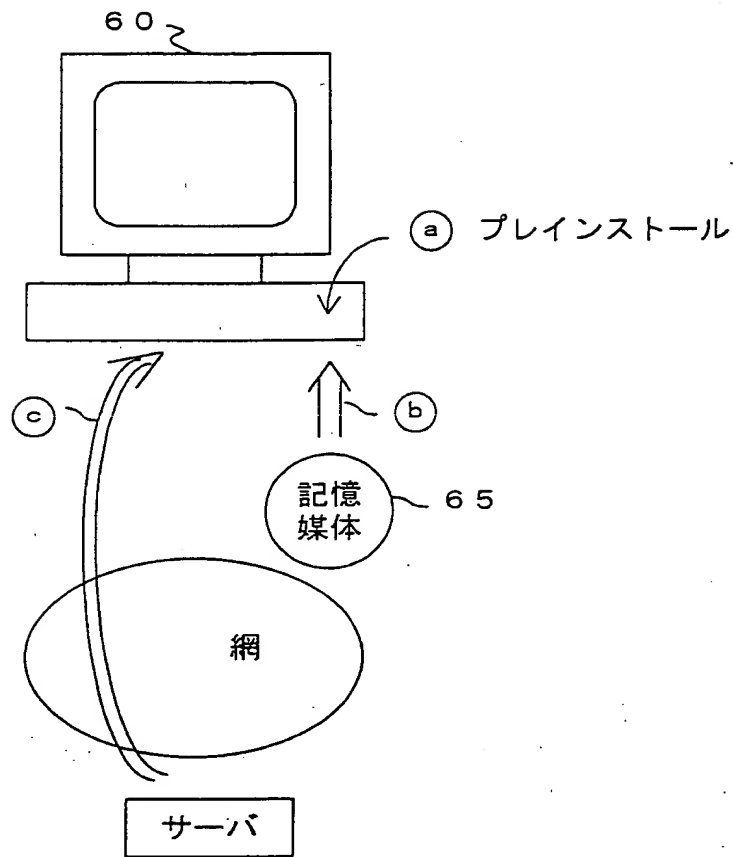


図 30

この謄本は原本と相違ないことを認証する。  
平成 13 年 3 月 12 日

経済産業事務官 清 寺 貴 幸

